BYTE

April–June 1987

LISTINGS SUPPLEMENT

INCLUDING THE

1986 EDITORIAL INDEX

Six great reasons to join BIX today

- **Over 140 *microcomputer-related conferences*:**
Join only those subjects that interest you and change selections at any time. Take part when it's convenient for you. Share information, opinions and ideas in focused discussions with other BIX users who share your interests. Easy commands and conference digests help you quickly locate important information.

- ***Monthly conference specials*:**
BIX specials connect you with invited experts in leading-edge topics—CD-ROM, MIDI, OS-9 and more. They're all part of your BIX membership.

- ***Microbytes daily*:**
Get up-to-the-minute industry news and new product information by joining Microbytes Daily and What's New Hardware and Software.

- ***Public domain software*:**
Yours for the downloading, including programs from BYTE articles and a growing library of PD listings.

- ***Electronic mail*:**
Exchange private messages with BYTE editors and authors and other BIX users.

- ***Vendor support*:**
A growing number of microcomputer manufacturers use BIX to answer your questions about their products and how to use them for peak performance.

What BIX Costs. . How You Pay

ONE-TIME REGISTRATION FEE: $25

Hourly Charges: (Your Time of Access)	Off-Peak 6PM–7AM Weekdays Plus Weekends & Holidays	Peak 7AM–6PM Weekdays
BIX	$9	$12
Tymnet*	$2	$6
TOTAL	**$11/hr.**	**$18/hr.****

* *Continental U.S. BIX is accessible via Tymnet from throughout the U.S. at charges much less than regular long distance. Call the BIX helpline number listed below for the Tymnet number near you or Tymnet at 1-800-336-0149.*
** *User is billed for time on system (i.e., ½ Hr. Off-Peak w/Tymnet = $5.50 charge).*

BIX and Tymnet charges billed by Visa or Mastercard only.

BIX HELPLINE
(8:30 AM–11:30 PM Eastern Weekdays)
U.S. (except NH)—1-800-227-BYTE
Elsewhere (603) 924-7681

We'll Send You a BIX *User's Manual and Subscriber Agreement as Soon as We've Processed Your Registration.*
JOIN THE EXCITING WORLD OF BIX TODAY!

JOIN BIX RIGHT NOW:
Set your computer's telecommunications program for full duplex, 8-bit characters, even parity, 1 stop bit OR 7-bit characters, even parity, 1 stop using 300 or 1200 baud.
Call your local Tymnet number and respond as follows:

Tymnet Prompt	You Enter
Garble or "terminal identifier"	a
login:	bytenet <CR>
password:	nigh <CR>
mhis login:	bix <CR>
BIX Logo—Name:	new <CR>

After you register on-line, you're immediately taken to the BIX learn conference and can start using the system right away.

FOREIGN ACCESS:
To access BIX from foreign countries, you must have an account with your local Postal Telephone & Telegraph (PTT) company. From your PTT enter 31065057878. Then enter bix <CR> and new <CR> at the prompts. Call or write us for PTT contact information.

BIX
One Phoenix Mill Lane
Peterborough, NH 03458
(603) 924-7681

TABLE OF CONTENTS

April 5

May 55

June 153

1986 Editorial Index 183

WELCOME TO BYTE'S QUARTERLY LISTINGS SUPPLEMENT

The BYTE Listings Supplement is produced quarterly as a means of providing interested readers with a printed, source code version of those programs referenced in BYTE articles. It provides a far more extensive look into the techniques of coding and the potentialities of microcomputers than we have space for in each month's BYTE.

Programs contained in this Supplement are referenced by the month the article appeared, the page on which their supporting article begins, and the name of the author who wrote the article.

For those who prefer programs already in electronic format, we have a companion service called Listings on Disk. If you have a modem, listings may be downloaded from the BYTEnet bulletin board and, if you are a member of BIX, the "Listings" area also contains programs referenced in BYTE.

If you live outside of the U.S., we've included the names, addresses and telephone numbers of bulletin boards that get program code from us. You'll find the directory just inside the back cover of this Supplement.

The bulletin boards are updated monthly. Several countries have enough boards that the telephone charges for most callers should be the minimum possible.

BYTE

BIX

Officers of McGraw-Hill Information Systems Company: President: Richard B. Miller. Executive Vice Presidents: Frederick P. Jannott, Construction Information Group; Russell C. White, Computers and Communications Information Group; J. Thomas Ryan, Marketing and International. Senior Vice Presidents: Francis A. Shinal, Controller; Robert C. Violette, Manufacturing and Technology. Senior Vice Presidents and Publishers: Laurence Altman, Electronics; Harry L. Brown, BYTE; David J. McGrath, Construction Publications. Group Vice President: Peter B. McCuen, Communications Information. Vice President: Fred O. Jensen, Planning and Development.

Officers of McGraw-Hill, Inc.: Harold W. McGraw, Jr., Chairman; Joseph L. Dionne, President and Chief Executive Officer; Robert N. Landes, Executive Vice President and Secretary; Walter D. Serwatka, Executive Vice President and Chief Financial Officer; Shel F. Asen, Senior Vice President, Manufacturing; Robert J. Bahash, Senior Vice President, Finance and Manufacturing; Ralph R. Schulz, Senior Vice President, Editorial; George R. Elsinger, Vice President, Circulation; Ralph J. Webb, Vice President and Treasurer.

BYTE, BYTE, and The Small Systems Journal are registered trademarks of McGraw-Hill Inc.

EDITORIAL AND BUSINESS OFFICE: One Phoenix Mill Lane, Peterborough, New Hampshire 03458, (603) 924-9281.

West Coast Offices: 425 Battery St., San Francisco, CA 94111, (415) 954-9718; 3001 Red Hill Ave., Building #1, Suite 222, Costa Mesa, CA 92626, (714) 557-6292. **New York Editorial Office:** 1221 Avenue of the Americas, New York, NY 10020, (212) 512-3175.

BYTEnet: (617) 861-9764 (set modem at 8-1-N or 7-1-E; 300 or 1200 baud).

BYTE (ISSN 0360-5280) is published monthly with additional issues in June and October by McGraw-Hill Inc. Founder: James H. McGraw (1860–1948). Executive, editorial, circulation, and advertising offices: One Phoenix Mill Lane, Peterborough, NH 03458, phone (603) 924-9281. Office hours: Monday through Thursday 8:30 AM–4:30 PM, Friday 8:30 AM–1:00 PM, Eastern Time. Address subscriptions to BYTE Subscriptions, P.O. Box 590, Martinsville, NJ 08836. Postmaster: send address changes, USPS Form 3579, undeliverable copies, and fulfillment questions to BYTE Subscriptions, P.O. Box 596, Martinsville, NJ 08836. Second-class postage paid at Peterborough, NH 03458 and additional mailing offices. Postage paid at Winnipeg, Manitoba. Registration number 9321. Subscriptions are $22 for one year, $40 for two years, and $58 for three years in the U.S. and its possessions. In Canada and Mexico, $25 for one year, $45 for two years, $65 for three years. $69 for one year air delivery to Europe. 31,000 yen for one year air delivery to Japan, 15,600 yen for one year surface delivery to Japan, $37 surface delivery elsewhere. Air delivery to selected areas at additional rates upon request. Single copy price is $3.50 in the U.S. and its possessions, $4.25 in Canada and Mexico, $4.50 in Europe, and $5 elsewhere. Foreign subscriptions and sales should be remitted in U.S. funds drawn on a U.S. bank. Please allow six to eight weeks for delivery of first issue. Printed in the United States of America.

Address editorial correspondence to: Editor, BYTE, One Phoenix Mill Lane, Peterborough, NH 03458. Unacceptable manuscripts will be returned if accompanied by sufficient postage. Not responsible for lost manuscripts or photos. Opinions expressed by the authors are not necessarily those of BYTE.

INDEX

April

386	TXT	5
BOYER	LSP	16
BROWSE	LSP	23
DERIV	LSP	26
DESTRUCT	LSP	26
DIV	LSP	27
EXE	LSP	28
FRPOLY	LSP	29
IO	LSP	31
LISTING	BIX	12
LOOP	LSP	32
PUZZLE	LSP	36
READ	ME	16
SIEVE	ASM	6
SIEVE386	ASM	9
TAK	LSP	38
TIMING	LSP	48
TRAVERSE	LSP	50
TRIANG	LSP	52

May

ADVAVL	H	56
ADVAVL	C	87
ADVCOM	H	55
ADVCOM	C	59
ADVDBS	H	57
ADVDBS	C	99
ADVEXE	C	109
ADVEXP	C	79
ADVFCN	C	67
ADVFIO	C	89
ADVINT	H	90
ADVINT	C	106
ADVJUNK	C	108
ADVMSG	C	91
ADVPRS	C	95
ADVSCN	C	75
ADVSYS	DOC	114
ADVTRM	C	92
LIST1	PS	147
LIST2	PS	147
LIST3	PS	148
LIST4	PS	148
LIST5	PS	149
LISTINGS	DOC	135
OBJECTS	ADI	128
OSAMPLE	ADV	126
PULLDOWN	C	136
README	1ST	55

June

BENCH	OBJ	180
CLOCKSET	ASM	156
COMPRESS	PAS	177
EXPAND	PAS	177
LISTING1	BAS	153
LISTING2	BAS	154
LISTING3	BAS	155
LISTING4	BAS	156
POLYFIT	BAS	178
TIMESET	ASM	171

1986 Index

..... 183

April

386.TXT Contributed by: Rick Grehan
TEXT from "Reviewer's Notebook," April 1987, page 201. Explanatory text, including a discussion of benchmarks developed by Rick Grehan.

HIGH C FROM METAWARE AND 80386 SOFTWARE DEVELOPMENT SERIES FROM PHAR LAP

The High C compiler from MetaWare, Inc. (903 Pacific Ave., Suite 201, Santa Cruz, CA 95060-4429) is now in release 1.3 that produces code for execution on an 80386 in protected mode. I tested the compiler on a Compaq Deskpro 386 running Compaq's DOS version 3.10.

High C programs may be configured to generate a variety of memory models: small, compact, medium, big, and large. On the low end, the small model requires that the code generated be less than 64K bytes (the code segment register remains fixed) - likewise, the program's data (including stack and heap) must fit within a 64K-byte segment. The large model permits the code segment register to be mobile, so that executable code may be greater than 64K bytes. Also, the DS (data segment) and ES (extra segment) registers are dynamic, permitting greater than 64K worth of data and heap space (only the stack is limited to 64K). The models between small and large incorporate combinations of static and dynamic code and data segments, so that from this selection, you should be able to choose a model that best suits your application. (The 80386 version of the compiler we tested currently only incorporates a version of the small memory model.)

The list of Hich C's features are quite extensive, and include:

* The ability to generate source code that can be tailored for use with assemblers.

* Provides extensive compiler checking normally found in "lint" programs. Also includes type checking as specified in ANSI C.

* Includes support for the 8087 and the 80287 math coprocessor chips. Special routines in the compiler detects the presence of a math coprocessor and sets a toggle that can be used to control compilation.

* There are two object files provided with the High C compiler that, when linked with your program, generate post mortem dumps. One file contains code to dump the contents of the heap in the event of an error condition - the other will display the call-chain of currently active functions.

* High C programs can make use of 'pragmas' - statements for controlling compiler parameters. For example, you can specify a directory search path, alter or re-instate compiler switches, perform conditional source-file includes, and more.

Documentation for High C consists of a thick three-ring binder divided into three general sections: a programmer's guide, a library reference manual, and a language reference manual. Each section is well indexed, and there appear to be a fair number of code fragments included as examples. (The software we received came with a demonstration .BAT file that compiled and executed three example programs.)

To execute code in protected mode, we used Phar Lap Software's 386LINK and RUN386 programs. Execution time was so small that it was obvious the Compaq was spending more time loading the programs than executing them.

386LINK and RUN386 are part of Phar Lap Software, Inc.'s (60 Aberdeen Ave., Cambridge, MA 02138) 80386 Software Development Series. This package consists of the 386ASM assembler, the 386LINK linker, the MINIBUG debugger, and RUN386 runtime environment. Versions of this package are available for the IBM PC, IBM PC/AT, VAX/VMS and a number of UNIX systems; the version we tested was running on a Compaq 386 Deskpro running Compaq's DOS version 3.10.

386ASM is a full-featured macro assembler with enough option switches to allow generation of code for practically every Intel processor since the 8088 (8088, 8086, 80186, 80286, and 80386). You can also assemble instructions for either the 80287 or 8087 numeric coprocessors. In operation, 386LINK acts much like any other linker, combining object files to create an MS-DOS .EXE file (you can optionally output the program in Intel hex file format).

continued

The MINIBUG debugger allows debugging of 80386 programs in either real mode or protected mode. It can be run on MS-DOS, can manage up to 4 breakpoints, and boasts commands similar to the standard MS-DOS/PC-DOS DEBUG program. Additonally, MINIBUG incorporates an online help screen that is invoked by entering a questionmark.

RUN386 creates an 80386 protected-mode runtime environment within MS-DOS. This means that programmers may work within the familiar MS-DOS world, creating applications that make full use of the 80386's power plus enjoy access to the standard MS-DOS system calls. (Some system calls are not supported: specifically, those involving memory allocation and interrupt vector manipulation.) RUN386 initializes necessary descriptors, loads the application into memory, switches into protected mode, and passes control to the application program. RUN386 will also transfer command line arguments to the application, and intercept all hardware interrupts to pass them along to standard interrupt handlers.

We rewrote the C version of the sieve program into 80386 assembler and ran it through 386ASM, 386LINK, and RUN386. Assemble time was less than 4 seconds, link time was less than 2, and execute time was so short that in order to make measurements we had to increase the number of iterations from 10 to 50. We then measured the program's execution time at 3.5 seconds.

Phar Lap's 386ASM manual was in second draft when we received it, and weighed in at 238 pages with no index (we sincerely hope they include one). The 386LINK, MINIBUG, and RUN386 manuals were in better shape - being 62, 54, and 21 pages, respectively (and with indices).

TABLE 1

	Compile/Link Time (sec)	File Size (bytes)	Execute Time (sec)
Sieve	14/21	34896	<1
Sort	15/21	30888	1

Table 1. Peformance benchmarks for High C compiler. The generated code was executed using Phar Lap Software, Inc.'s RUN386 program that allows 80386 protected-mode programs to run within an MS-DOS environment.

SIEVE.ASM Contributed by: Rick Grehan
TEXT "Reviewer's Notebook," April 1987, page 201. This is one of the benchmarks developed by Rick Grehan.

```
; 80386 assembly version of sieve program.
; Prints out the number of primes found in hexadecimal.
; (Printing out the number of primes found in decimal
;  is left as an exercise for the reader.)
;
; First define constants
ITERVAL         equu    50      ;Iteration count
TRUE            equu    1
FALSE           equu    0
ASIZE           equu    8190    ;Array size

                assume cs:sieve,ds:sdata
sieve           segment para public use32 'code'
                public _start_

;
; *** ENTRY POINT ***

_start_         proc    near
;
; Tell user we are beginning the program
;
                lea     ebx,startmsg
                call    printmsg
```

```
;
; Setup loop counter
                mov     ax,ITERVAL              ;Number of iterations
                mov     iter,ax
;
; Initialize counter for number of primes
L0:             xor     eax,eax
                mov     count,ax
;
; Set all flags true
                lea     ebx,flags
                mov     ecx,ASIZE+1
L1:             mov     byte ptr[ebx],TRUE
                inc     ebx
                loop    L1
;
; Primary loop
                mov     index,eax
L2:             mov     ebx,index
                mov     al,flags[ebx]
                or      al,al                   ;Is it a prime?
                jz      L5
;
; Found a prime
                mov     eax,ebx                 ;Twice index plus 3
                add     eax,ebx
                add     eax,3
;
; Kill all multiples
L3:             add     ebx,eax
                cmp     ebx,ASIZE
                jg      L4                  mov     byte ptr flags[ebx],FALSE
                jmp     L3
;
; Count number of primes
L4:             inc     word ptr count
;
L5:             inc     dword ptr index
                cmp     dword ptr index,ASIZE+1
                jne     L2
;
; Do another iteration
                dec     word ptr iter
                jnz     L0
;
; Print out number of primes
; For now, in hex
                lea     ebx,npmsg
                call    printmsg
                mov     ax,count
                call    hwout
;
; Terminate process
                mov     ax,04C00h
                int     21H

_start_         endp
;
; THE FOLLOWING CODE WAS INCLUDED WITH THE PHAR LAP 386ASM/
; 386LINK PACKAGE AS PART OF DEMO SOFTWARE.
;
;       printmsg - Print a message to the screen
;
;       ebx - Points to the message to be printed out.  The message
;             must be null terminated.
;

printmsg proc   near

        push    edx                     ; Save EDX.

pm1:    mov     dl,[ebx]                ; Load the next character into DL and
        or      dl,dl                   ;    branch if null.
        je      pm3                     ;
```

continued

```
        mov     ah,02h                  ; Output the character.
        int     21h                     ;

        cmp     byte ptr [ebx],0ah      ; If the character is a LF, then
        jne     pm2                     ;    also output a CR.
        mov     ah,02h                  ;
        mov     dl,0dh                  ;
        int     21h                     ;

pm2:    add     ebx,1                   ; Increment the message pointer and
        jmp     pm1                     ;    loop.
pm3:    pop     edx                     ; Restore EDX and return.
        ret                             ;

printmsg endp                           ;

;
;       hwout - Output the hex word in AX to the screen
;
hwout   proc    near                    ;

        push    ax                      ; Output the high byte of the word.
        ror     ax,8                    ;
        call    hbout                   ;

        pop     ax                      ; Output the low byte of the word.
        call    hbout                   ;

        ret                             ; Return.

hwout   endp                            ;


;
;       hbout - Output the hex byte in AL to the screen
;
hbout   proc    near                    ;

        push    ax                      ; Output the high digit of the byte.
        ror     ax,4                    ;
        call    hdout                   ;

        pop     ax                      ; Output the low digit of the byte.
        call    hdout                   ;

        ret                             ; Return.

hbout   endp


;
;       hdout - Output the hex digit in AL to the screen
;
hdout   proc    near

        and     ax,0fh                  ; Zap any extra bits and translate
        cmp     ax,10                   ;    to ASCII.
        jg      hd1                     ;
        add     al,'0'-'A'+10           ;
hd1:    add     al,'A'-10               ;

        push    dx                      ; Call MS-DOS to output the digit.
        mov     dl,al                   ;       mov     ah,2h                   ;
        int     21h                     ;
        pop     dx                      ;

        ret                             ; Return.

hdout   endp


sieve           ends
```

```
sdata           segment para public use32 'data'

startmsg        db      'Beginning sieve.',0dh,0ah,0
npmsg           db      'Number of primes:',0

flags           db      ASIZE+1 dup (?)
index           dd      ?
iter            dw      ?
count           dw      ?

sdata           ends


_stack          segment byte stack use32 'stack'
                db      8000 dup (?)

_stack          ends
                end
```

SIEVE386.ASM Contributed by: Rick Grehan
TEXT "Reviewer's Notebook," April 1987, page 201. Another of the benchmarks developed by Rick Grehan.

```
; 80386 assembly version of sieve program.
; Prints out the number of primes found in hexadecimal.
; (Printing out the number of primes found in decimal
;  is left as an exercise for the reader.)
;
; First define constants
ITERVAL         equ     50      ;Iteration count
TRUE            equ     1
FALSE           equ     0
ASIZE           equ     8190    ;Array size

                assume cs:sieve,ds:sdata
sieve           segment para public use32 'code'
                public _start_

;
; *** ENTRY POINT ***

_start_         proc    near
;
; Tell user we are beginning the program;
                lea     ebx,startmsg
                call    printmsg
;
; Setup loop counter
                mov     ax,ITERVAL              ;Number of iterations
                mov     iter,ax
;
; Initialize counter for number of primes
L0:             xor     eax,eax
                mov     count,ax
;
; Set all flags true
                lea     ebx,flags
                mov     ecx,ASIZE+1
L1:             mov     byte ptr[ebx],TRUE
                inc     ebx
                loop    L1
;
; Primary loop
                mov     index,eax
L2:             mov     ebx,index
                mov     al,flags[ebx]
                or      al,al                   ;Is it a prime?
                jz      L5
```

continued

```
;
; Found a prime
                mov     eax,ebx                 ;Twice index plus 3
                add     eax,ebx
                add     eax,3
;
; Kill all multiples
L3:             add     ebx,eax
                cmp     ebx,ASIZE
                jg      L4
                mov     byte ptr flags[ebx],FALSE
                jmp     L3
;
; Count number of primes
L4:             inc     word ptr count
;
L5:             inc     dword ptr index
                cmp     dword ptr index,ASIZE+1
                jne     L2
;
; Do another iteration
                dec     word ptr iter
                jnz     L0
;
; Print out number of primes
; For now, in hex
                lea     ebx,npmsg
                call    printmsg
                mov     ax,count
                call    hwout;
; Terminate process
                mov     ax,04C00h
                int     21H

_start_         endp
;
; THE FOLLOWING CODE WAS INCLUDED WITH THE PHAR LAP 386ASM/
; 386LINK PACKAGE AS PART OF DEMO SOFTWARE.
;
;       printmsg - Print a message to the screen
;
;       ebx - Points to the message to be printed out.  The message
;             must be null terminated.
;

printmsg proc   near

        push    edx                     ; Save EDX.

pm1:    mov     dl,[ebx]                ; Load the next character into DL and
        or      dl,dl                   ;   branch if null.
        je      pm3                     ;

        mov     ah,02h                  ; Output the character.
        int     21h                     ;

        cmp     byte ptr [ebx],0ah      ; If the character is a LF, then
        jne     pm2                     ;   also output a CR.
        mov     ah,02h                  ;
        mov     dl,0dh                  ;
        int     21h                     ;

pm2:    add     ebx,1                   ; Increment the message pointer and
        jmp     pm1                     ;   loop.

pm3:    pop     edx                     ; Restore EDX and return.
        ret                             ;

printmsg endp                           ;

;
;       hwout - Output the hex word in AX to the screen
;
```

```
hwout   proc    near                    ;

        push    ax                      ; Output the high byte of the word.
        ror     ax,8                    ;
        call    hbout                   ;

        pop     ax                      ; Output the low byte of the word.
        call    hbout                   ;

        ret                             ; Return.
hwout   endp                            ;


;
;       hbout - Output the hex byte in AL to the screen
;

hbout   proc    near                    ;

        push    ax                      ; Output the high digit of the byte.
        ror     ax,4                    ;
        call    hdout                   ;

        pop     ax                      ; Output the low digit of the byte.
        call    hdout                   ;

        ret                             ; Return.

hbout   endp


;
;       hdout - Output the hex digit in AL to the screen
;

hdout   proc    near

        and     ax,0fh                  ; Zap any extra bits and translate
        cmp     ax,10                   ;    to ASCII.
        jg      hd1                     ;
        add     al,'0'-'A'+10           ;
hd1:    add     al,'A'-10               ;

        push    dx                      ; Call MS-DOS to output the digit.
        mov     dl,al                   ;
        mov     ah,2h                   ;
        int     21h                     ;
        pop     dx                      ;

        ret                             ; Return.

hdout   endp

sieve           ends


sdata           segment para public use32 'data'

startmsg        db      'Beginning sieve.',0dh,0ah,0
npmsg           db      'Number of primes:',0

flags           db      ASIZE+1 dup (?)
index           dd      ?
iter            dw      ?count          dw      ?

sdata           ends


_stack          segment byte stack use32 'stack'
                db      8000 dup (?)

_stack          ends
                end
```

continued

LISTING.BIX Contributed by: Mukkai S. Krishnamoorthy and Snorri Agnarsson
TEXT Programming Project: "Concurrent Programming in Turbo Pascal," by Mukkai S. Krishnamoorthy and Snorri Agnarsson, April, 1987. All the files associated with this article.

```
{ NAME:  newprocess
  EXAMPLE CALL:
    p:=NewProcess(Ofs(proc),1000);
       proc is the parameterless procedure, from which
       the new process is created.   The stack of the
       new process p is 1000 bytes.
}
function NewProcess(prog: integer; size: integer): Process;
var stack: ^integer;
begin
GetMem(stack,size);
MemW[Seg(stack^):Ofs(stack^)+size-10]:=prog;
MemW[Seg(stack^):Ofs(stack^)+size-12]:=Ofs(stack^)+size-12;
NewProcess:=Ptr(Seg(stack^),Ofs(stack^)+size-12);
end;
```

[Listing 1.]

```
; procedure transfer(var p1,p2: Process);
;
cseg        segment 'cgroup'
            assume  cs:cseg
transfer    proc    near
;
  push    bp        ;   Turbo Pascal generated prolog
  mov     bp,sp     ;     -         -         -       -
;
  pop     bp              ; Align with 'newprocess' setup
  les     bp,dword ptr [bp]+4  ; get address of p2
  mov     ax,es:[bp]+2       ; get segment part of p2
  mov     bx,es:[bp]         ; get offset part of p2
  mov     bp,sp                ; bp - point to parm's
  les     bp,dword ptr [bp]+8  ; get address of p1
  mov     es:[bp],sp         ; store sp in offset part
  mov     es:[bp]+2,ss       ; store ss in segment part
  mov     ss,ax              ; new stack segment from p2
  mov     sp,bx              ; new stack pointer from p2
  mov     bp,sp              ; re-establish bp for epilog
;
   mov     sp,bp           ; Turbo Pascal generated epilog
   pop     bp          ;       -       -       -       -
   ret     8           ;       -       -       -       -
;
transfer    endp
cseg        ends
```

[Listing 2a]

```
procedure transfer(var p1,p2: process);
begin
inline(
$5D/ $C4/ $6E/ $04/ $26/ $8B/ $46/ $02/ $26/ $8B/ $5E/ $00/
$8B/ $EC/ $C4/ $6E/ $08/ $26/ $89/ $66/ $00/ $26/ $8C/ $56/
$02/ $8E/ $D0/ $8B/ $E3/ $8B/ $EC);
end;
```

[Listing 2b]

```
cseg        segment 'cgroup'
     assume  cs:cseg
inthandler   proc    near
  jmp    start ; jump over data area
```

```
getbase:
  call    base ; subroutine to get base of data area.
base:
  pop     di   ; pop address of base into di.
  ret          ; return with offset of base in di.
;  data area:
newdsword    dw   ?  ; data segment register for pascal
stkoffset    dw   ?  ; offset of stack
stksegment   dw   ?  ; segment of stack for pascal
procoffset   dw   ?  ; offset of interrupt handler procedure
                     ; segment of handler must be callsegment
calloffset   dw   ?  ; offset of routine that makes short call
callsegment  dw   ?  ; segment of routine that makes short call
savessword   dw   ?  ; word to save ss into
savespword   dw   ?  ; word to save sp into
newds        equ  newdsword-base     ; offset from base to newdsword
newsp        equ  stkoffset-base     ; offset from base to stkoffset
newss        equ  stksegment-base    ; offset from base to stksegment
handler      equ  procoffset-base    ; offset from base to procoffset
caller       equ  calloffset-base    ; offset from base to calloffset
savess       equ  savessword-base    ; offset from base to savessword
savesp       equ  savespword-base    ; offset from base to savespword
start:
  push    di                         ; save di
  call    getbase                    ; get base of data area in di
  mov     word ptr cs:[di]+savess,ss   ; save ss
  mov     word ptr cs:[di]+savesp,sp   ; save sp
  mov     ss,word ptr cs:[di]+newss    ; get new ss
  mov     sp,word ptr cs:[di]+newsp    ; get new sp
  push    ax                           ; save the rest of the registers
  push    bx
  push    cx
  push    dx
  push    bp
  push    si
  push    es
  push    ds
  mov     ds,word ptr cs:[di]+newds    ; get ds for pascal
  mov     bx,word ptr cs:[di]+handler ; get offset of handler
  call    dword ptr cs:[di]+caller     ; long call to short caller
  pop     ds                           ; restore all registers
  pop     es                           ; and return from interrupt
  pop     si
  pop     bp
  pop     dx
  pop     cx
  pop     bx
  pop     ax
  call    getbase
  mov     ss,word ptr cs:[di]+savess
  mov     sp,word ptr cs:[di]+savesp
  pop     di
  iret
inthandler   endp
cseg         ends
```

[Listing 3]

```
 cseg         segment 'cgroup'
              assume  cs:cseg
 shortcaller  proc    far
              call    bx
              ret
 shortcaller  endp
 cseg         ends
```

[Listing 4]

```
{  NAME:  newioprocess
   EXAMPLE CALL:
    p:=NewIoProcess(Ofs(proc),1000);
   proc is the parameterless procedure, from which
   the new ioprocess is created. The stack of the
   new ioprocess p is 1000 bytes.
```

continued

```
}
function newioprocess(proc: integer; size: integer): ioprocess;
procedure shortcaller;
begin
inline($FF/$D3/$CB);
end;
const inthandler: array[1..85] of byte=
(
$EB, $16, $90, $E8, $00, $00, $5F, $C3, $00, $00, $00, $00,
$00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00, $00,
$57, $E8, $E7, $FF, $2E, $8C, $55, $0E, $2E, $89, $65, $10,
$2E, $8E, $55, $06, $2E, $8B, $65, $04, $50, $53, $51, $52,
$55, $56, $06, $1E, $2E, $8E, $5D, $02, $2E, $8B, $5D, $08,
$2E, $FF, $5D, $0A, $1F, $07, $5E, $5D, $5A, $59, $5B, $58,
$E8, $B8, $FF, $2E, $8E, $55, $0E, $2E, $8B, $65, $10, $5F,
$CF);
var area: ^integer;
begin
GetMem(area,size+85);
Move(inthandler,area^,85);
memw[Seg(area^):Ofs(area^)+ 8]:=Dseg;
memw[Seg(area^):Ofs(area^)+10]:=Ofs(area^)+size+85;
memw[Seg(area^):Ofs(area^)+12]:=Seg(area^);
memw[Seg(area^):Ofs(area^)+14]:=proc;
memw[Seg(area^):Ofs(area^)+16]:=Ofs(shortcaller)+12;
memw[Seg(area^):Ofs(area^)+18]:=Cseg;
newioprocess:=area;
end;
```

[Listing 5]

```
{  NAME:  IoAttach
   PARAMETERS:
    'intnum' is an interrupt number
    'proc' is an ioprocess created by newioprocess
}
procedure IoAttach(intnum: byte; proc: ioprocess);
var regs: record
            ax,bx,cx,dx,bp,si,di,ds,es,flags: integer
          end;
begin
with regs do
   begin
   ax:=$2500 + intnum;  { DOS function 25H sets an }
   ds:=Seg(proc^);      { interrupt vector.        }
   dx:=Ofs(proc^);
   end;
MsDos(regs);       { request DOS function }
end;
```

[Listing 6]

```
{$K-} { turn off checking for stack overflow }

program multitest;

type Process=^integer;

    ...  { definitions of NewProcess & transfer }

var p1,p2: process;

procedure prog1;
begin
while true do   begin
  writeln('Hi');
  transfer(p1,p2);
  writeln('He');
  transfer(p1,p2);
  end;
end;
```

```
procedure prog2;
begin
while true do
  begin
  writeln('Ho');
  transfer(p2,p1);
  end;
end;

var p0: process;

procedure main;
begin
p1:=newprocess(ofs(prog1),1000);
p2:=newprocess(ofs(prog2),1000);
transfer(p0,p1);
end;

 begin main end.
```

[Listing 7a]

Resulting output:

```
  Hi
  Ho
  He
  Ho
  Hi
  Ho
  .
  .
  .
```

[Listing 7b]

```
{$K-}  { turn of checking for stack overflow }

program interrupttest;
type IoProcess = ^integer;

var count: integer;
var timerhandler: IoProcess;

... { definitions of NewIoProcess and IoAttach }

procedure incrementer;
begin
count:=succ(count);
end;

begin
timerhandler:=NewIoProcess(Ofs(incrementer),1000);
count:=0;
IoAttach($1C,timerhandler); { attach timerhandler to user }
while true do               { timer interrupt ( 1Ch )     }
   begin
   writeln(count);
   Delay(100);              { delay 100 milliseconds      }
   end;
end.
```

continued

[Listing 8a]

Resulting output:

```
0
1
3
5
7
8
10
12
.
.
.
```

[Listing 8b]

READ.ME Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

These benchmarks are for Gold Hill Common LISP 286 Developer.
This LISP implementation runs on an IBM PC AT; the interpreter
requires at least 1.5 megabytes of memory; the compiler requires
at least 3 megabytes of memory. You should first load the file
TIMING.LSP. This file contains functions for loading, compiling,
and running the benchmarks.

BOYER.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; BOYER

(defvar unify-subst)
(defvar temp-temp)

(DEFUN ADD-LEMMA (TERM)
  (COND ((AND (NOT (ATOM TERM))
              (EQ (CAR TERM)
                  (QUOTE EQUAL))
              (NOT (ATOM (CADR TERM))))
         ;; This change lets you run setup several times.
         (unless (member term (get (car (cadr term)) 'lemmas) :test #'equal)
           (SETF (GET (CAR (CADR TERM)) (QUOTE LEMMAS))
                 (CONS TERM (GET (CAR (CADR TERM)) (QUOTE LEMMAS))))))
        (T
         (error "Add lemma did not like term")
         )))

(DEFUN ADD-LEMMA-LST (LST)
       (COND ((NULL LST)
              T)
             (T (ADD-LEMMA (CAR LST))
                (ADD-LEMMA-LST (CDR LST)))))

(DEFUN APPLY-SUBST (ALIST TERM)
       (COND ((ATOM TERM)
              (COND ((SETQ TEMP-TEMP (ASSOC TERM ALIST))
                     (CDR TEMP-TEMP))
                    (T TERM)))
             (T (CONS (CAR TERM)
                      (APPLY-SUBST-LST ALIST (CDR TERM))))))
```

```
(DEFUN APPLY-SUBST-LST (ALIST LST)
       (COND ((NULL LST)
              NIL)
             (T (CONS (APPLY-SUBST ALIST (CAR LST))
                      (APPLY-SUBST-LST ALIST (CDR LST))))))

(DEFUN FALSEP (X LST)
       (OR (EQUAL X (QUOTE (F)))
           (MEMBER X LST)))

(DEFUN ONE-WAY-UNIFY (TERM1 TERM2)
       (PROGN (SETQ UNIFY-SUBST NIL)
              (ONE-WAY-UNIFY1 TERM1 TERM2)))

(DEFUN ONE-WAY-UNIFY1 (TERM1 TERM2)
       (COND ((ATOM TERM2)
              (COND ((SETQ TEMP-TEMP (ASSOC TERM2 UNIFY-SUBST))
                     (EQUAL TERM1 (CDR TEMP-TEMP)))
                    (T (SETQ UNIFY-SUBST (CONS (CONS TERM2 TERM1)
                                               UNIFY-SUBST))
                       T)))
             ((ATOM TERM1)
              NIL)
             ((EQ (CAR TERM1)
                  (CAR TERM2))
              (ONE-WAY-UNIFY1-LST (CDR TERM1)
                                  (CDR TERM2)))
             (T NIL)))

(DEFUN ONE-WAY-UNIFY1-LST (LST1 LST2)
       (COND ((NULL LST1)
              T)
             ((ONE-WAY-UNIFY1 (CAR LST1)
                              (CAR LST2))
              (ONE-WAY-UNIFY1-LST (CDR LST1)
                                  (CDR LST2)))
             (T NIL)))

(DEFUN REWRITE (TERM)
       (COND ((ATOM TERM)
              TERM)
             (T (REWRITE-WITH-LEMMAS (CONS (CAR TERM)
                                           (REWRITE-ARGS (CDR TERM)))
                                     (GET (CAR TERM)
                                          (QUOTE LEMMAS))))))

(DEFUN REWRITE-ARGS (LST)
       (COND ((NULL LST)
              NIL)
             (T (CONS (REWRITE (CAR LST))
                      (REWRITE-ARGS (CDR LST))))))

(DEFUN REWRITE-WITH-LEMMAS (TERM LST)
       (COND ((NULL LST)
              TERM)
             ((ONE-WAY-UNIFY TERM (CADR (CAR LST)))
              (REWRITE (APPLY-SUBST UNIFY-SUBST (CADDR (CAR LST)))))
             (T (REWRITE-WITH-LEMMAS TERM (CDR LST)))))

(DEFUN SETUP ()
  (ADD-LEMMA-LST
    (QUOTE ((EQUAL (COMPILE FORM)
                   (REVERSE (CODEGEN (OPTIMIZE FORM)
                                     (NIL))))
            (EQUAL (EQP X Y)
                   (EQUAL (FIX X)
                          (FIX Y)))
            (EQUAL (GREATERP X Y)
                   (LESSP Y X))
            (EQUAL (LESSEQP X Y)
                   (NOT (LESSP Y X)))
            (EQUAL (GREATEREQP X Y)
                   (NOT (LESSP X Y)))
```

continued

```
(EQUAL (BOOLEAN X)
       (OR (EQUAL X (T))
           (EQUAL X (F))))
(EQUAL (IFF X Y)
       (AND (IMPLIES X Y)
            (IMPLIES Y X)))
(EQUAL (EVEN1 X)
       (IF (ZEROP X)
           (T)
           (ODD (1- X))))
(EQUAL (COUNTPS- L PRED)
       (COUNTPS-LOOP L PRED (ZERO)))
(EQUAL (FACT- I)
       (FACT-LOOP I 1))
(EQUAL (REVERSE- X)
       (REVERSE-LOOP X (NIL)))
(EQUAL (DIVIDES X Y)
       (ZEROP (remainder Y X)))
(EQUAL (ASSUME-TRUE VAR ALIST)
       (CONS (CONS VAR (T))
             ALIST))
(EQUAL (ASSUME-FALSE VAR ALIST)
       (CONS (CONS VAR (F))
             ALIST))
(EQUAL (TAUTOLOGY-CHECKER X)
       (TAUTOLOGYP (NORMALIZE X)
                   (NIL)))
(EQUAL (FALSIFY X)
       (FALSIFY1 (NORMALIZE X)
                 (NIL)))
(EQUAL (PRIME X)
       (AND (NOT (ZEROP X))
            (NOT (EQUAL X (ADD1 (ZERO))))
            (PRIME1 X (1- X))))
(EQUAL (AND P Q)
       (IF P (IF Q (T)
                   (F))
           (F)))
(EQUAL (OR P Q)
       (IF P (T)
           (IF Q (T)
                 (F))
           (F)))
(EQUAL (NOT P)
       (IF P (F)
           (T)))
(EQUAL (IMPLIES P Q)
       (IF P (IF Q (T)
                   (F))
           (T)))
(EQUAL (FIX X)
       (IF (NUMBERP X)
           X
           (ZERO)))
(EQUAL (IF (IF A B C)
           D E)
       (IF A (IF B D E)
           (IF C D E)))
(EQUAL (ZEROP X)
       (OR (EQUAL X (ZERO))
           (NOT (NUMBERP X))))
(EQUAL (PLUS (PLUS X Y)
             Z)
       (PLUS X (PLUS Y Z)))
(EQUAL (EQUAL (PLUS A B)
              (ZERO))
       (AND (ZEROP A)
            (ZEROP B)))
(EQUAL (DIFFERENCE X X)
       (ZERO))
(EQUAL (EQUAL (PLUS A B)
              (PLUS A C))
       (EQUAL (FIX B)
              (FIX C)))
```

```
(EQUAL (EQUAL (ZERO)
              (DIFFERENCE X Y))
       (NOT (LESSP Y X)))
(EQUAL (EQUAL X (DIFFERENCE X Y))
       (AND (NUMBERP X)
            (OR (EQUAL X (ZERO))
                (ZEROP Y))))
(EQUAL (MEANING (PLUS-TREE (APPEND X Y))
                A)
       (PLUS (MEANING (PLUS-TREE X)
                      A)
             (MEANING (PLUS-TREE Y)
                      A)))
(EQUAL (MEANING (PLUS-TREE (PLUS-FRINGE X))
                A)
       (FIX (MEANING X A)))
(EQUAL (APPEND (APPEND X Y)
               Z)
       (APPEND X (APPEND Y Z)))
(EQUAL (REVERSE (APPEND A B))
       (APPEND (REVERSE B)
               (REVERSE A)))
(EQUAL (TIMES X (PLUS Y Z))
       (PLUS (TIMES X Y)
             (TIMES X Z)))
(EQUAL (TIMES (TIMES X Y)
              Z)
       (TIMES X (TIMES Y Z)))
(EQUAL (EQUAL (TIMES X Y)
              (ZERO))
       (OR (ZEROP X)
           (ZEROP Y)))
(EQUAL (EXEC (APPEND X Y)
             PDS ENVRN)
       (EXEC Y (EXEC X PDS ENVRN)
             ENVRN))
(EQUAL (MC-FLATTEN X Y)
       (APPEND (FLATTEN X)
               Y))
(EQUAL (MEMBER X (APPEND A B))
       (OR (MEMBER X A)
           (MEMBER X B)))
(EQUAL (MEMBER X (REVERSE Y))
       (MEMBER X Y))
(EQUAL (LENGTH (REVERSE X))
       (LENGTH X))
(EQUAL (MEMBER A (INTERSECT B C))
       (AND (MEMBER A B)
            (MEMBER A C)))
(EQUAL (NTH (ZERO)
            I)
       (ZERO))
(EQUAL (EXP I (PLUS J K))
       (TIMES (EXP I J)
              (EXP I K)))
(EQUAL (EXP I (TIMES J K))
       (EXP (EXP I J)
            K))
(EQUAL (REVERSE-LOOP X Y)
       (APPEND (REVERSE X)
               Y))
(EQUAL (REVERSE-LOOP X (NIL))
       (REVERSE X))
(EQUAL (COUNT-LIST Z (SORT-LP X Y))
       (PLUS (COUNT-LIST Z X)
             (COUNT-LIST Z Y)))
(EQUAL (EQUAL (APPEND A B)
              (APPEND A C))
       (EQUAL B C))
(EQUAL (PLUS (REMAINDER X Y)
             (TIMES Y (QUOTIENT X Y)))
       (FIX X))
```

continued

```
(EQUAL (POWER-EVAL (BIG-PLUS1 L I BASE)
                   BASE)
       (PLUS (POWER-EVAL L BASE)
             I))
(EQUAL (POWER-EVAL (BIG-PLUS X Y I BASE)
                   BASE)
       (PLUS I (PLUS (POWER-EVAL X BASE)
                     (POWER-EVAL Y BASE))))
(EQUAL (REMAINDER Y 1)
       (ZERO))
(EQUAL (LESSP (REMAINDER X Y)
              Y)
       (NOT (ZEROP Y)))
(EQUAL (REMAINDER X X)
       (ZERO))
(EQUAL (LESSP (QUOTIENT I J)
              I)
       (AND (NOT (ZEROP I))
            (OR (ZEROP J)
                (NOT (EQUAL J 1)))))
(EQUAL (LESSP (REMAINDER X Y)
              X)
       (AND (NOT (ZEROP Y))
            (NOT (ZEROP X))
            (NOT (LESSP X Y))))
(EQUAL (POWER-EVAL (POWER-REP I BASE)
                   BASE)
       (FIX I))
(EQUAL (POWER-EVAL (BIG-PLUS (POWER-REP I BASE)
                             (POWER-REP J BASE)
                             (ZERO)
                             BASE)
                   BASE)
       (PLUS I J))
(EQUAL (GCD X Y)
       (GCD Y X))
(EQUAL (NTH (APPEND A B)
            I)
       (APPEND (NTH A I)
               (NTH B (DIFFERENCE I (LENGTH A)))))
(EQUAL (DIFFERENCE (PLUS X Y)
                   X)
       (FIX Y))
(EQUAL (DIFFERENCE (PLUS Y X)
                   X)
       (FIX Y))
(EQUAL (DIFFERENCE (PLUS X Y)
                   (PLUS X Z))
       (DIFFERENCE Y Z))
(EQUAL (TIMES X (DIFFERENCE C W))
       (DIFFERENCE (TIMES C X)
                   (TIMES W X)))
(EQUAL (REMAINDER (TIMES X Z)
                  Z)
       (ZERO))
(EQUAL (DIFFERENCE (PLUS B (PLUS A C))
                   A)
       (PLUS B C))
(EQUAL (DIFFERENCE (ADD1 (PLUS Y Z))
                   Z)
       (ADD1 Y))
(EQUAL (LESSP (PLUS X Y)
              (PLUS X Z))
       (LESSP Y Z))
(EQUAL (LESSP (TIMES X Z)
              (TIMES Y Z))
       (AND (NOT (ZEROP Z))
            (LESSP X Y)))
(EQUAL (LESSP Y (PLUS X Y))
       (NOT (ZEROP X)))
(EQUAL (GCD (TIMES X Z)
            (TIMES Y Z))
       (TIMES Z (GCD X Y)))
(EQUAL (VALUE (NORMALIZE X)
              A)
       (VALUE X A))
```

```
(EQUAL (EQUAL (FLATTEN X)
              (CONS Y (NIL)))
       (AND (NLISTP X)
            (EQUAL X Y)))
(EQUAL (LISTP (GOPHER X))
       (LISTP X))
(EQUAL (SAMEFRINGE X Y)
       (EQUAL (FLATTEN X)
              (FLATTEN Y)))
(EQUAL (EQUAL (GREATEST-FACTOR X Y)
              (ZERO))
       (AND (OR (ZEROP Y)
                (EQUAL Y 1))
            (EQUAL X (ZERO))))
(EQUAL (EQUAL (GREATEST-FACTOR X Y)
              1)
       (EQUAL X 1))
(EQUAL (NUMBERP (GREATEST-FACTOR X Y))
       (NOT (AND (OR (ZEROP Y)
                     (EQUAL Y 1))
                 (NOT (NUMBERP X)))))
(EQUAL (TIMES-LIST (APPEND X Y))
       (TIMES (TIMES-LIST X)
              (TIMES-LIST Y)))
(EQUAL (PRIME-LIST (APPEND X Y))
       (AND (PRIME-LIST X)
            (PRIME-LIST Y)))
(EQUAL (EQUAL Z (TIMES W Z))
       (AND (NUMBERP Z)
            (OR (EQUAL Z (ZERO))
                (EQUAL W 1))))
(EQUAL (GREATEREQPR X Y)
       (NOT (LESSP X Y)))
(EQUAL (EQUAL X (TIMES X Y))
       (OR (EQUAL X (ZERO))
           (AND (NUMBERP X)
                (EQUAL Y 1))))
(EQUAL (REMAINDER (TIMES Y X)
                  Y)
       (ZERO))
(EQUAL (EQUAL (TIMES A B)
              1)
       (AND (NOT (EQUAL A (ZERO)))
            (NOT (EQUAL B (ZERO)))
            (NUMBERP A)
            (NUMBERP B)
            (EQUAL (1- A)
                   (ZERO))
            (EQUAL (1- B)
                   (ZERO))))
(EQUAL (LESSP (LENGTH (DELETE X L))
              (LENGTH L))
       (MEMBER X L))
(EQUAL (SORT2 (DELETE X L))
       (DELETE X (SORT2 L)))
(EQUAL (DSORT X)
       (SORT2 X))
(EQUAL (LENGTH (CONS X1
                     (CONS X2
                           (CONS X3 (CONS X4
                                          (CONS X5
                                                (CONS X6 X7)))))))
       (PLUS 6 (LENGTH X7)))
(EQUAL (DIFFERENCE (ADD1 (ADD1 X))
                   2)
       (FIX X))
(EQUAL (QUOTIENT (PLUS X (PLUS X Y))
                 2)
       (PLUS X (QUOTIENT Y 2)))
(EQUAL (SIGMA (ZERO)
              I)
       (QUOTIENT (TIMES I (ADD1 I))
                 2))
```

continued

```
            (EQUAL (PLUS X (ADD1 Y))
                   (IF (NUMBERP Y)
                       (ADD1 (PLUS X Y))
                       (ADD1 X)))
            (EQUAL (EQUAL (DIFFERENCE X Y)
                          (DIFFERENCE Z Y))
                   (IF (LESSP X Y)
                       (NOT (LESSP Y Z))
                       (IF (LESSP Z Y)
                           (NOT (LESSP Y X))
                           (EQUAL (FIX X)
                                  (FIX Z)))))
            (EQUAL (MEANING (PLUS-TREE (DELETE X Y))
                            A)
                   (IF (MEMBER X Y)
                       (DIFFERENCE (MEANING (PLUS-TREE Y)
                                            A)
                                   (MEANING X A))
                       (MEANING (PLUS-TREE Y)
                                A)))
            (EQUAL (TIMES X (ADD1 Y))
                   (IF (NUMBERP Y)
                       (PLUS X (TIMES X Y))
                       (FIX X)))
            (EQUAL (NTH (NIL)
                        I)
                   (IF (ZEROP I)
                       (NIL)
                       (ZERO)))
            (EQUAL (LAST (APPEND A B))
                   (IF (LISTP B)
                       (LAST B)
                       (IF (LISTP A)
                           (CONS (CAR (LAST A))                    B)
                           B)))
            (EQUAL (EQUAL (LESSP X Y)
                          Z)
                   (IF (LESSP X Y)
                       (EQUAL T Z)
                       (EQUAL F Z)))
            (EQUAL (ASSIGNMENT X (APPEND A B))
                   (IF (ASSIGNEDP X A)
                       (ASSIGNMENT X A)
                       (ASSIGNMENT X B)))
            (EQUAL (CAR (GOPHER X))
                   (IF (LISTP X)
                       (CAR (FLATTEN X))
                       (ZERO)))
            (EQUAL (FLATTEN (CDR (GOPHER X)))
                   (IF (LISTP X)
                       (CDR (FLATTEN X))
                       (CONS (ZERO)
                             (NIL))))
            (EQUAL (QUOTIENT (TIMES Y X)
                             Y)
                   (IF (ZEROP Y)
                       (ZERO)
                       (FIX X)))
            (EQUAL (GET J (SET I VAL MEM))
                   (IF (EQP J I)
                       VAL
                       (GET J MEM)))))

(DEFUN TAUTOLOGYP (X TRUE-LST FALSE-LST)
       (COND ((TRUEP X TRUE-LST)
              T)
             ((FALSEP X FALSE-LST)
              NIL)
             ((ATOM X)
              NIL)
             ((EQ (CAR X)
                  (QUOTE IF))
              (COND ((TRUEP (CADR X)
                            TRUE-LST)
                     (TAUTOLOGYP (CADDR X)
                                 TRUE-LST FALSE-LST))
```

```
                ((FALSEP (CADR X)
                         FALSE-LST)
                 (TAUTOLOGYP (CAr (cDDDR X))
                             TRUE-LST FALSE-LST))
                (T (AND (TAUTOLOGYP (CADDR X)
                                    (CONS (CADR X)
                                          TRUE-LST)
                                    FALSE-LST)
                        (TAUTOLOGYP (CAr (cDDDR X))
                                    TRUE-LST
                                    (CONS (CADR X)
                                          FALSE-LST))))))
          (T NIL)))

(DEFUN TAUTP (X)
       (TAUTOLOGYP (REWRITE X)
                   NIL NIL))
(DEFUN TEST NIL
       (PROG (ANS TERM)
             (SETQ TERM
                   (APPLY-SUBST
                     (QUOTE ((X F (PLUS (PLUS A B)
                                        (PLUS C (ZERO))))
                              (Y F (TIMES (TIMES A B)
                                          (PLUS C D)))
                              (Z F (REVERSE (APPEND (APPEND A B)
                                                    (NIL))))
                              (U EQUAL (PLUS A B)
                                 (DIFFERENCE X Y))
                              (W LESSP (REMAINDER A B)
                                 (MEMBER A (LENGTH B)))))
                      (QUOTE (IMPLIES (AND (IMPLIES X Y)
                                           (AND (IMPLIES Y Z)
                                                (AND (IMPLIES Z U)
                                                     (IMPLIES U W))))
                                      (IMPLIES X W)))))
             (SETQ ANS (TAUTP TERM))))

(DEFUN TRANS-OF-IMPLIES (N)
       (LIST (QUOTE IMPLIES)
             (TRANS-OF-IMPLIES1 N)
             (LIST (QUOTE IMPLIES)
                   0 N)))

(DEFUN TRANS-OF-IMPLIES1 (N)
       (COND ((EQUAL N 1)
              (LIST (QUOTE IMPLIES)
                    0 1))
             (T (LIST (QUOTE AND)
                      (LIST (QUOTE IMPLIES)
                            (1- N)
                            N)
                      (TRANS-OF-IMPLIES1 (1- N))))))
(DEFUN TRUEP (X LST)
       (OR (EQUAL X (QUOTE (T)))
           (MEMBER X LST)))

(SETUP)

(define-timer boyer "Boyer" (test))

(qa-attempt "boyer" (test) nil)
```

BROWSE.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; BROWSE
; Benchmark to create and browse through an AI-like data base of units
```

continued

```
(defvar rand 21.)

#-GCLisp
(defmacro char1 (x) '(aref (symbol-name ,x) 0))

#+GCLisp ; Hack, hack.  Don't cons up strings to get first char!
(defmacro char1 (x)
  '(multiple-value-bind (.off. .seg.) (sys::%pointer ,x)
     (sys::%contents .seg. (+& .off. 21))))
(defun init (n m npats ipats)
  (let ((ipats (copy-tree ipats)))
    (do ((p ipats (cdr p)))
        ((null (cdr p)) (rplacd p ipats)))
    (do ((n n (1- n))
         (i m (cond ((= i 0) m)
                    (t (1- i))))
         (name (gensym) (gensym))
         (a ()))
        ((= n 0) a)
      (push name a)
      (do ((i i (1- i)))
          ((= i 0))
        (setf (get name (gensym)) ()))
      (setf (get name 'pattern)
            (do ((i npats (1- i))
                 (ipats ipats (cdr ipats))
                 (a ()))
                ((= i 0) a)
              (push (car ipats) a)))
      (do ((j (- m i) (1- j)))
          ((= j 0))
        (setf (get name (gensym) ) ())))))

(defun browse-random () (setq rand (mod (* rand 17.) 251.)))

(defun randomize (l)
  (do ((a ()))
      ((null l) a)
    (let ((n (mod (browse-random) (length l))))
      (cond ((= n 0)
             (push (car l) a)
             (setq l (cdr l)))
            (t
             (do ((n n (1- n))
                  (x l (cdr x)))
                 ((= n 1)
                  (push (cadr x) a)
                  (rplacd x (cddr x)))))))))
(defun match (pat dat alist)
  (cond ((null pat)
         (null dat))
        ((null dat) ())
        ((or (eq (car pat) '?)
             (eq (car pat)
                 (car dat)))
         (match (cdr pat) (cdr dat) alist))
        ((eq (car pat) '*)
         (or (match (cdr pat) dat alist)
             (match (cdr pat) (cdr dat) alist)
             (match pat (cdr dat) alist)))
        (t (cond ((atom (car pat))
                  (cond ((eq (char1 (car pat)) #\?)     ; long story
                         (let ((val (assoc (car pat) alist)))
                           (cond (val (match (cons (cdr val)
                                                   (cdr pat))
                                             dat alist))
                                 (t (match (cdr pat)
                                           (cdr dat)
                                           (cons (cons (car pat)
                                                       (car dat))
                                                 alist))))))
                        ((eq (char1 (car pat)) #\*)
                         (let ((val (assoc (car pat) alist)))
                           (cond (val (match (append (cdr val)
                                                     (cdr pat))
                                             dat alist))
```

```
                                (t
                                 (do ((l () (nconc l (list (car d))))
                                      (e (cons () dat) (cdr e))
                                      (d dat (cdr d)))
                                     ((null e) ())
                                   (cond ((match (cdr pat) d
                                                 (cons (cons (car pat) l)
                                                       alist))
                                          (return t)))))))))
                    (t (and
                         (not (atom (car dat)))
                         (match (car pat)
                                (car dat) alist)
                         (match (cdr pat)
                                (cdr dat) alist)))))))
(defun browse ()
  (setf rand 21)
  (investigate (randomize
                 (init 100. 10. 4. '((a a a b b b b a a a a a b b a a a)
                                     (a a b b b b a a
                                        (a a)(b b))
                                     (a a a b (b a) b a b a))))
               '((*a ?b *b ?b a *a a *b *a)
                 (*a *b *b *a (*a) (*b))
                 (? ? * (b a) * ? ?))))
(defun investigate (units pats)
  (do ((units units (cdr units)))
      ((null units))
    (do ((pats pats (cdr pats)))
        ((null pats))
      (do ((p (get (car units) 'pattern)
              (cdr p)))
          ((null p))
        (match (car pats) (car p) ()))))))

(define-timer browse "Browse" (browse))

(qa-attempt "Browse" (browse) nil)
```

DERIV.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; DERIV

(DEFUN deriv-aux (A) (LIST '/ (DERIV A) A))

(DEFUN DERIV (A)
  (COND
    ((ATOM A)
     (COND ((EQ A 'X) 1) (T 0)))
    ((EQ (CAR A) '+)
     (CONS '+ (MAPCAR #'DERIV (CDR A))))
    ((EQ (CAR A) '-)
     (CONS '- (MAPCAR #'DERIV (CDR A))))
    ((EQ (CAR A) '*)
     (LIST '*
           A
           (CONS '+ (MAPCAR 'deriv-aux (CDR A)))))
    ((EQ (CAR A) '/)
     (LIST '-
           (LIST '/
                 (DERIV (CADR A))
                 (CADDR A))
           (LIST '/
                 (CADR A)
                 (LIST '*
                       (CADDR A)
                       (CADDR A)
                       (DERIV (CADDR A))))))
```

continued

```
    (T 'ERROR)))

(DEFUN RUN-deriv ()
  (DO ((I 0 (1+ I)))
      ((= I 1000.))
    #-GCLisp (DECLARE (type FIXNUM I))
    (DERIV '(+ (* 3 X X) (* A X X) (* B X) 5))
    (DERIV '(+ (* 3 X X) (* A X X) (* B X) 5))    (DERIV '(+ (* 3 X X) (* A X X) (* B X) 5))
    (DERIV '(+ (* 3 X X) (* A X X) (* B X) 5))
    (DERIV '(+ (* 3 X X) (* A X X) (* B X) 5))))

(define-timer deriv "Deriv" (run-deriv))
(qa-attempt "Deriv" (run-deriv) nil)


;;; 3.11 DDERIV

(DEFUN dderiv-aux (A) (LIST '/ (DDERIV A) A))

(DEFUN +DDERIV (A)
  (CONS '+ (MAPCAR #'DDERIV A)))

(DEFUN -DDERIV (A)
  (CONS '- (MAPCAR #'DDERIV A)))

(DEFUN *DDERIV (A)
       (LIST '* (CONS '* A)
                (CONS '+ (MAPCAR #'dderiv-aux A))))

(DEFUN /DDERIV (A)
       (LIST '-
             (LIST '/
                   (DDERIV (CAR A))
                   (CADR A))
             (LIST '/
                   (CAR A)
                   (LIST '*
                         (CADR A)
                         (CADR A)
                         (DDERIV (CADR A))))))

(DEFUN DDERIV (A)
  (COND
    ((ATOM A)
     (COND ((EQ A 'X) 1) (T 0)))
    (T (LET ((DDERIV (GET (CAR A) 'DDERIV)))
         (COND (DDERIV (FUNCALL DDERIV (CDR A)))
               (T 'ERROR))))))

(defun setup-dderiv ()
  (mapc #'(lambda (op fun)
            (setf (get op 'dderiv) (symbol-function fun)))
        '(+ - * /)
        '(+dderiv -dderiv *dderiv /dderiv)))

(setup-dderiv)

(DEFUN RUN-dderiv ()
  (DO ((I 0 (1+ I)))
      ((= I 1000.))
    #-GCLisp (DECLARE (type FIXNUM I))
    (DDERIV '(+ (* 3 X X) (* A X X) (* B X) 5))    (DDERIV '(+ (* 3 X X) (* A X X) (* B X) 5))
    (DDERIV '(+ (* 3 X X) (* A X X) (* B X) 5))
    (DDERIV '(+ (* 3 X X) (* A X X) (* B X) 5))
    (DDERIV '(+ (* 3 X X) (* A X X) (* B X) 5))))

(define-timer dderiv "DDeriv" (run-dderiv))
(qa-attempt "DDeriv" (run-dderiv) nil)
```

DESTRUCT.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; DESTRUCT
; Destructive operation benchmark

(defun destructive (n m)
  (let ((l (do ((i 10. (1- i))
                (a () (push () a)))
               ((= i 0) a))))
    (do ((l n (1- l)))
        ((= l 0))
      (cond ((null (car l))
             (do ((l l (cdr l)))
         ((null l))
               (or (car l)
                   (rplaca l (cons () ())))
               (nconc (car l)
                      (do ((j m (1- j))
                           (a () (push () a)))
                          ((= j 0) a)))))
            (t
             (do ((l1 l (cdr l1))
                  (l2 (cdr l) (cdr l2)))
                 ((null l2))
               (rplacd (do ((j (floor (length (car l2)) 2) (1- j))
                            (a (car l2) (cdr a)))
                           ((= j 0) a)
                         (rplaca a l))
                       (let ((n (floor (length (car l1)) 2)))
                         (cond ((= n 0) (rplaca l1 ())
                                (car l1))
                               (t
                                (do ((j n (1- j))
                                     (a (car l1) (cdr a)))
                                    ((= j 1)
                                     (prog1 (cdr a)
                                            (rplacd a ())))
                                  (rplaca a l)))))))))))

(define-timer destruct "Destruct" (destructive 600. 50.))

(qa-attempt "Destruct" (destructive 600. 50.) nil)
```

DIV.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; DIV2
; Dividing by 2 using lists of n ()'s

(defun create-n (n)
       (do ((n n (1- n))
            (a () (push () a)))
           ((= n 0) a)))

(defvar div2-l )
(setq div2-l (create-n 200.))

(defun iterative-div2 (l)
  (do ((l l (cddr l))
       (a () (push (car l) a)))
      ((null l) a)))
```

continued

```
(defun recursive-div2 (l)
  (cond ((null l) ())
        (t (cons (car l) (recursive-div2 (cddr l))))))

(defun iterative-div2-test (l)
       (do ((i 300. (1- i)))
           ((= i 0))
           (iterative-div2 l)
           (iterative-div2 l)
           (iterative-div2 l)
           (iterative-div2 l)))

(defun recursive-div2-test (l)
       (do ((i 300. (1- i)))
           ((= i 0))
           (recursive-div2 l)
           (recursive-div2 l)
           (recursive-div2 l)
           (recursive-div2 l)))

(define-timer div2-1 "Div2, Iterative" (iterative-div2-test div2-l))
(qa-attempt "Div2, Iterative" (iterative-div2-test div2-l) nil)

(define-timer div2-2 "Div2, Recursive" (recursive-div2-test div2-l))
(qa-attempt "Div2, Recursive" (recursive-div2-test div2-l) nil)
```

EXE.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
(defun show-exe (file)
  (setq file (merge-pathnames file ".EXE"))
  (with-open-file (stream file :direction :input :element-type 'unsigned-byte)
    (format t "&%EXE file header of ")
    (show-pathname file)
    (format t ":%%")
    (let ((signature (show-dbyte "Link signature" t stream)))
      (if (= signature #x5A4D) (format t "  (correct)")
   (format t "  (incorrect)")))
    (let ((leftover (show-dbyte "Image length mod 512" nil stream))
   (pages (show-dbyte "Image length/512" nil stream)))
      (format t "&Image length: D" (+ (* pages 512) leftover)))
    (show-dbyte "Relocation table length" nil stream)
    (show-dbyte "Header size (paragraphs)" nil stream)
    (show-dbyte "Minimum extra memory (paragraphs)" nil stream)
    (show-dbyte "Maximum extra memory (paragraphs)" nil stream)
    (show-dbyte "Stack segment offset" t stream)
    (show-dbyte "Initial SP" t stream)
    (show-dbyte "Checksum" t stream)
    (show-dbyte "Initial IP" t stream)
    (show-dbyte "Code segment offset" t stream)
    (show-dbyte "Relocation table offset" nil stream)
    (show-dbyte "Overlay number" nil stream)))

(defun show-dbyte (what hex? stream)
  (let* ((b1 (read-byte stream))
  (b2 (read-byte stream))
  (dbyte (logior (ash b2 8) b1)))
    (format t "&A: " what)
    (if hex? (format t "Xh" dbyte) (format t "D" dbyte))
    dbyte))

(defun show-pathname (pathname)
  (format t "A\\" (pathname-device pathname))
  (let ((dirs (pathname-directory pathname)))
    (when (listp dirs)
      (setq dirs (rest dirs))
      (do () ((null dirs))
```

```
 (format t "A\\" (first dirs))
 (setq dirs (rest dirs))))
    (format t "A" (pathname-name pathname))
    (if (pathname-type pathname) (format t ".A" (pathname-type pathname)))))
```

FRPOLY.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; FRPOLY

(defvar *v*)
(defvar *X*)
(defvar *alpha*)
(defvar *A*)
(defvar *B*)
(defvar *B*)
(defvar *l)
(defvar *p)
(defvar q*)
(defvar u*)
(defvar *var)
(defvar *y*)
(defvar *R*)
(defvar *r2*)
(defvar *r3*)

;(declare (localf pcoefadd pcplus pcplus1 pplus ptimes ptimes1
;                 ptimes2 ptimes3 psimp pctimes pctimes1
;                 pplus1))

(defmacro pointergp (x y) '(> (get ,x 'order)(get ,y 'order)))
(defmacro pcoefp (e) '(atom ,e))

(defmacro pzerop (x) '(and (numberp ,x) (zerop ,x)))
                     ;true
for 0 or 0.0
;(defmacro pzero () 0)
(defmacro cplus (x y) '(+ ,x ,y))
(defmacro ctimes (x y) '(* ,x ,y))

(defun pcoefadd (e c x)
  (if (pzerop c)
      x
      (cons e (cons c x))))

(defun pcplus (c p)
  (if (pcoefp p)
      (cplus p c)
      (psimp (car p) (pcplus1 c (cdr p)))))

(defun pcplus1 (c x)
       (cond ((null x)
              (cond ((pzerop c) nil) (t (cons 0 (cons c nil)))))
             ((pzerop (car x)) (pcoefadd 0 (pplus c (cadr x)) nil))
             (t (cons (car x) (cons (cadr x) (pcplus1 c (cddr x)))))))

(defun pctimes (c p) (cond ((pcoefp p) (ctimes c p))
                           (t (psimp (car p) (pctimes1 c (cdr p))))))

(defun pctimes1 (c x)
       (cond ((null x) nil)
             (t (pcoefadd (car x)
                          (ptimes c (cadr x))
                          (pctimes1 c (cddr x))))))
```

continued

```
(defun pplus (x y) (cond ((pcoefp x) (pcplus x y))
                         ((pcoefp y) (pcplus y x))
                         ((eq (car x) (car y))
                          (psimp (car x) (pplus1 (cdr y) (cdr x))))
                         ((pointergp (car x) (car y))
                          (psimp (car x) (pcplus1 y (cdr x))))
                         (t (psimp (car y) (pcplus1 x (cdr y))))))

(defun pplus1 (x y)
       (cond ((null x) y)
             ((null y) x)
             ((= (car x) (car y))
              (pcoefadd (car x)
                        (pplus (cadr x) (cadr y))
                        (pplus1 (cddr x) (cddr y))))
             ((> (car x) (car y))
              (cons (car x) (cons (cadr x) (pplus1 (cddr x) y))))
             (t (cons (car y) (cons (cadr y) (pplus1 x (cddr y)))))))

(defun psimp (var x)
       (cond ((null x) 0)
             ((atom x) x)
             ((zerop (car x)) (cadr x))
              (t (cons var x))))

(defun ptimes (x y) (cond ((or (pzerop x) (pzerop y)) 0)
                          ((pcoefp x) (pctimes x y))
                          ((pcoefp y) (pctimes y x))
                          ((eq (car x) (car y))
                           (psimp (car x) (ptimes1 (cdr x) (cdr y))))
                          ((pointergp (car x) (car y))
                           (psimp (car x) (pctimes1 y (cdr x))))
                          (t (psimp (car y) (pctimes1 x (cdr y))))))

(defun ptimes1 (*x* y) (prog (u* *v*)
                             (setq *v* (setq u* (ptimes2 y)))
                           a (setq *x* (cddr *x*))
                             (cond ((null *x*) (return u*)))
                             (ptimes3 y)
                             (go a)))

(defun ptimes2 (y) (cond ((null y) nil)
                         (t (pcoefadd (+ (car *x*) (car y))
                                      (ptimes (cadr *x*) (cadr y))
                                      (ptimes2 (cddr y))))))

(defun ptimes3 (y)
  (prog (e u c)
    a1 (cond ((null y) (return nil)))
       (setq e (+ (car *x*) (car y)))
       (setq c (ptimes (cadr y) (cadr *x*) ))
       (cond ((pzerop c) (setq y (cddr y)) (go a1))
             ((or (null *v*) (> e (car *v*)))
              (setq u* (setq *v* (pplus1 u* (list e c))))
              (setq y (cddr y)) (go a1))
             ((= e (car *v*))
              (setq c (pplus c (cadr *v*)))
              (cond ((pzerop c) (setq u* (setq *v* (pdiffer1 u* (list (car*v*) (cadr *v*))))))
                    (t (rplaca (cdr *v*) c)))
              (setq y (cddr y))
              (go a1)))
     a (cond ((and (cddr *v*) (> (caddr *v*) e)) (setq *v* (cddr *v*)) (go a)))
       (setq u (cdr *v*))
     b (cond ((or (null (cdr u)) (< (cadr u) e))
              (rplacd u (cons e (cons c (cdr u)))) (go e)))
       (cond ((pzerop (setq c (pplus (caddr u) c))) (rplacd u (cdddr u)) (go d))
             (t (rplaca (cddr u) c)))
     e (setq u (cddr u))
     d (setq y (cddr y))
       (cond ((null y) (return nil)))
       (setq e (+ (car *x*) (car y)))
       (Setq c (ptimes (cadr y) (cadr *x*)))
     c (cond ((and (cdr u) (> (cadr u) e))
              (setq u (cddr u)) (go c)))
       (go b)))
```

```
;; pdiffer1 is referred to above but not defined.  RPG says it is never
called.
(defun pdiffer1 (x y) x y (error "pdiffer2 called"))

(defun pexptsq (p n)
       (do ((n (floor n 2) (floor n 2))
            (s (cond ((oddp n) p) (t 1))))
           ((zerop n) s)
           (setq p (ptimes p p))
           (and (oddp n) (setq s (ptimes s p))) ))

(defun setup-frpoly nil
  (setf (get 'x 'order ) 1)
  (setf (get 'y 'order ) 2)
  (setf (get 'z 'order ) 3)
  (setq *r* (pplus '(x 1 1 0 1) (pplus '(y 1 1) '(z 1 1)))) ; r= x+y+z+1
  (setq *r2* (ptimes *r* 100000)) ;r2 = 100000*r
  (setq *r3* (ptimes *r* 1.0)); r3 = r with floating point coefficients
  )

(setup-frpoly)

(define-timer frpoly2r "FRPoly, Power = 2, r = x + y + z + 1" (pexptsq *r* 2))
(define-timer frpoly2r2 "FRPoly, Power = 2, r2 = 1000r" (pexptsq *r2* 2))
(define-timer frpoly2r3 "FRPoly, Power = 2, r3 = r in flonums" (pexptsq *r3*2))

(qa-attempt "FRPoly, Power = 2, r = x + y + z + 1" (pexptsq *r* 2)
 (Z 2 1 1 (Y 1 2 0 (X 1 2 0 2)) 0 (Y 2 1 1 (X 1 2 0 2) 0 (X 2 1 1 3 0 1))))


(qa-attempt "FRPoly, Power = 2, r3 = r in flonums" (pexptsq *r3* 2)
 (Z 2 1.0 1 (Y 1 2.0 0 (X 1 2.0 0 2.0)) 0
         (Y 2 1.0 1 (X 1 2.0 0 2.0) 0 (X 2 1.0 1 3.0 0 1.0))))
(define-timer frpoly5r "FRPoly, Power = 5, r = x + y + z + 1" (pexptsq *r* 5))
(define-timer frpoly5r2 "FRPoly, Power = 5, r2 = 1000r" (pexptsq *r2* 5))
(define-timer frpoly5r3 "FRPoly, Power = 5, r3 = r in flonums" (pexptsq *r3*
5))

(define-timer frpoly10r "FRPoly, Power = 10, r = x + y + z + 1" (pexptsq *r*
10))
(define-timer frpoly10r2 "FRPoly, Power = 10, r2 = 1000r" (pexptsq *r2* 10))
(define-timer frpoly10r3 "FRPoly, Power = 10, r3 = r in flonums" (pexptsq *r3*
10))

(define-timer frpoly15r "FRPoly, Power = 15, r = x + y + z + 1" (pexptsq *r*
15))
(define-timer frpoly15r2 "FRPoly, Power = 15, r2 = 1000r" (pexptsq *r2* 15))
(define-timer frpoly15r3 "FRPoly, Power = 15, r3 = r in flonums" (pexptsq *r3*
15))
```

IO.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; FPRINT
; Benchmark to print to a file.

(defvar test-atoms '(abcdef12 cdefgh23 efghij34 ghijkl45 ijklmn56 klmnop67
                              mnopqr78 opqrst89 qrstuv90 stuvwx01 uvwxyz12
                              wxyzab23 xyzabc34 123456ab 234567bc 345678cd
                              456789de 567890ef 678901fg 789012gh 890123hi))

(defun fprint-init (m n atoms)
       (let ((atoms (copy-tree atoms)))
            (do ((a atoms (cdr a)))
                ((null (cdr a)) (rplacd a atoms)))
            (fprint-init1 m n atoms)))
```

continued

```
(defun fprint-init1 (m n atoms)
       (cond ((= m 0) (pop atoms))
             (t (do ((i n (- i 2))
                     (a ()))
                    ((< i 1) a)
                    (push (pop atoms) a)
                    (push (fprint-init1 (1- m) n atoms) a)))))

(defvar test-pattern (fprint-init 6. 6. test-atoms))

(defparameter fprint-test-file "FPRINT.TST")

(defun fprint ()
  (let ((f (open fprint-test-file :direction :output)))
    (print test-pattern f)
    (close f)))

(define-timer fprint "FPrint" (fprint))
; FREAD
; Benchmark to read from a file.

(defun fread ()
  (let ((f (open fprint-test-file)))
    (read f)
    (close f)))

(define-timer fread "FRead" (fread))

; TPRINT
; Benchmark to print and read to the terminal

(defvar test-atoms '(abc1 cde2 efg3 ghi4 ijk5 klm6 mno7 opq8 qrs9
                          stu0 uvw1 wxy2 xyz3 123a 234b 345c 456d
                          567d 678e 789f 890g))

(defun tprint-init (m n atoms)
       (let ((atoms (copy-tree atoms)))
            (do ((a atoms (cdr a)))
                ((null (cdr a)) (rplacd a atoms)))
            (tprint-init1 m n atoms)))

(defun tprint-init1 (m n atoms)
       (cond ((= m 0) (pop atoms))
             (t (do ((i n (- i 2))
                     (a ()))
                    ((< i 1) a)
                    (push (pop atoms) a)
                    (push (tprint-init1 (1- m) n atoms) a)))))

(defvar test-pattern (tprint-init 6. 6. test-atoms))

(define-timer tprint "TPrint" (print test-pattern))
```

LOOP.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; the LOOP Macro

(in-package 'SYS)

(defmacro loop (&body body)
  (if (symbolp (first body)) (loop-translate body)
      (let ((tag (gensym)))
 `(block nil (tagbody ,tag ,@body (go ,tag))))))

(defmacro l (&body body) (pprint (loop-translate body)) nil)
```

```
(defvar *loop-collect-keywords* '("APPEND" "APPENDING" "COLLECT" "COLLECTING"
      "NCONC" "NCONCING"))
(defvar *loop-keywords* '("APPEND" "APPENDING" "AS" "COLLECT" "COLLECTING"
     "DO" "DOING" "FINALLY" "FOR" "IF" "INITIALLY"
                            "NAMED" "NCONC" "NCONCING" "UNLESS" "UNTIL"
     "WHEN" "WHILE" "WITH"))

(defun loop-keyword? (object)
  (and (symbolp object)
       (member (string object) *loop-keywords* :test #'string-equal)))

(defmacro loop-finish () '(go loop-exit-tag))

(defun add-loop-bindings (bindings variable value)
  (setf (first bindings)
        (nconc (first bindings)
               (cond ((not (listp variable))
                      (list (list variable value)))
                     ((relatively-atomic value)
                      (generate-loop-destructuring variable value))
                     (t (let ((temp (gensym)))
                          (add-loop-bindings (rest bindings) temp value)
                          (generate-loop-destructuring variable temp)))))))

(defun relatively-atomic (form)
  (or (symbolp form)
      (and (member (first form) '(car cdr caar cadr cdar cddr caaar caadr
                                  cadar caddr cdaar cdadr cddar cdddr))
           (relatively-atomic (second form))
           (null (cddr form)))))

(defun generate-loop-destructuring (variables values)
  (cond ((null variables) ())
        ((atom variables) (list (list variables values)))
        (t (nconc (generate-loop-destructuring
                    (car variables) (if (null values) nil '(car ,values)))
                  (generate-loop-destructuring
                    (cdr variables) (if (null values) nil '(cdr ,values)))))))

(defun add-for-bindings (bindings forms variable value)
  (nconc forms
         (cond ((not (listp variable)) '((setf ,variable ,value)))
               ((relatively-atomic value)
                (list (generate-for-destructuring variable value)))
               (t (let ((temp (gensym)))
                    (add-loop-bindings bindings temp nil)
                    (list '(setf ,temp ,value)
                          (generate-for-destructuring variable temp)))))))

(defun generate-for-destructuring (variable value)
  (let ((bindings (generate-loop-destructuring variable value)))
    (if (= (length bindings) 1) (cons 'setf (first bindings))
        (cons 'psetf (apply #'nconc bindings)))))

(eval-when (eval compile)
(defmacro lppop (x)  '(if (null ,x) (error "LOOP expression terminates unexpectedly.") (pop
,x))))

(defun loop-collect-form (key symbol expression)
  (setf key (aref key 0))
  (cond ((char-equal key #\C) ; COLLECT
         '(nconc ,symbol (list ,expression)))
        ((char-equal key #\A) ; APPEND
         '(append ,symbol ,expression))
        (t ; NCONC
         '(nconc ,symbol ,expression))))


(defun loop-for-translate (bindings preset-forms reset-forms body for?)
  (let ((key (lppop body)) (temp nil) (temp2 nil) (var nil))
    (tagbody
```

continued

```
next    (unless (symbolp key) (go set))
        (when (loop-keyword? key) (go exit))
        (when (string-equal (string key) "AND")
          (setf key (lppop body))
          (setf temp (string key))
          (if (string-equal temp "FOR") (setf for? 't)
              (if (string-equal temp "AS") (setf for? nil)))
          (go next))
set     (setf var key)
        (setf key (lppop body))
        (unless (symbolp key)
          (add-loop-bindings bindings var nil) (go next))
        (setf temp (string key))
        (when (string-equal temp "AND")
          (add-loop-bindings bindings var nil) (go next))
        (when (loop-keyword? temp)
          (add-loop-bindings bindings var nil) (go exit))
        (cond ((string-equal temp "=")  ;; "FOR/AS X ="
               (setf key (lppop body))
               (add-loop-bindings bindings var key)
               (unless for? ;; "AS X ="
                 (setf reset-forms
                       (add-for-bindings bindings reset-forms var key))
                 (setf key (lppop body)) (go next))
               (setf key (lppop body))
               (unless (and (symbolp key)
                            (string-equal (string key) "THEN"))
                 (go next)) ;; "FOR X = Y THEN"
               (setf key (lppop body))
               (setf reset-forms
                     (add-for-bindings bindings reset-forms var key))
               (setf key (lppop body))
               (go next))
              ((member temp '("FROM" "DOWNFROM" "UPFROM")
                       :test #'string-equal)
               (unless for? (error "Bad LOOP phrase: AS S A" var temp))
               (let ((by (cond ((string-equal temp "UPFROM") 1)
                               ((string-equal temp "DOWNFROM") -1)
                               (t nil))))                          (setf key (lppop body))
                 (add-loop-bindings bindings var key)
                 (setf key (lppop body))
                 (unless (symbolp key)
                   (setf reset-forms
                         (add-for-bindings bindings reset-forms var
                                           `(+ ,var ,(or by 1))))
                   (go next))
                 (setf temp2 (string key))
                 (setf key (lppop body))
                 (when (string-equal temp2 "BY")
                   (when by (error "Ill-formed LOOP FOR: S A BY ..."
                                   var temp))
                   (setf reset-forms
                         (add-for-bindings bindings reset-forms var
                                           `(+ ,var ,key)))
                   (go next))
                 (unless (member temp2 '("TO" "DOWNTO" "UPTO"
                                         "BELOW" "ABOVE")
                                 :test #'string-equal)

                   (setf reset-forms
                         (add-for-bindings bindings reset-forms var
                                           `(+ ,var ,(or by 1))))
                   (go next))
                 (BREAK)))
              ((string-equal temp "IN")
               (setf key (lppop body))
               (setf temp (gensym))
               (add-loop-bindings bindings temp key)
               (setf preset-forms
                     (nconc preset-forms
                            `((if (null ,temp) (loop-finish)))))
               (setf preset-forms
                     (add-for-bindings bindings preset-forms var `(car,temp)))
```

```
                (setf key (lppop body))
                (cond ((and (symbolp key) (string-equal (string key) "BY"))
                       (setf key (lppop body))
                       (setf reset-forms
                             (add-for-bindings bindings reset-forms temp
                                               '(funcall ,key ,temp)))
                       (setf key (lppop body)))
                      (t (setf reset-forms
                               (add-for-bindings bindings reset-forms temp
                                                 '(cdr ,temp)))))
                (go next))
               (t (error "FOR/AS keyword expected in LOOP expression: S"
                         key)))
      exit)
    (values preset-forms reset-forms body key)))

(defun loop-translate (body)
  (do ((name nil)                                ; Loop name.
       (bindings ())                             ; LET bindings to be made.       (forms ())
       (init-forms ())                           ; Loop initialization forms.
       (exit-forms ())                           ; Loop finish forms.
       (preset-forms ())                         ; Loop prepass var reset forms.
       (reset-forms ())                          ; Loop pass var reset forms.
       (key (lppop body))                        ; Next keyword to process.
       (temp nil))
      ((null body)
       (do ((answer '(tagbody ,@init-forms loop-enter-tag
                              ,@preset-forms ,@forms ,@reset-forms
                              (go loop-enter-tag)
                              loop-exit-tag ,@exit-forms)
                    (let ((binding (pop bindings)))
                      (if (null binding) answer
                          '(let ,binding ,answer)))))
           ((null bindings) '(block ,name ,answer))))
    (if (not (symbolp key))
        (error "Random form where LOOP keyword expected: S" key))
    (setf key (string key))
    (cond ((string-equal key "NAMED")
           (if name (error "LOOP body contains two NAMED keys."))
           (setf name (lppop body))
           (unless (symbolp name) (error "Bad LOOP name: S" name))
           (setf key (lppop body)))
          ((string-equal key "INITIALLY")
           (loop (setf key (lppop body))
                 (if (loop-keyword? key) (return nil))
                 (setf init-forms (nconc init-forms (list key)))
                 (unless body (return nil))))
          ((string-equal key "FINALLY")
           (loop (setf key (pop body))
                 (if (loop-keyword? key) (return nil))
                 (when (and (symbolp key)
                            (string-equal (string key) "RETURN"))
                   (setf exit-forms
                         (nconc exit-forms '((return ,(lppop body)))))
                   (setf key (lppop body))
                   (return nil))
                 (setf exit-forms (nconc exit-forms (list key)))
                 (unless body (return nil))))
          ((string-equal key "WHILE")
           (setf temp (lppop body))
           (setf key (lppop body))
           (setf forms (nconc forms '((unless ,temp (loop-finish))))))
          ((string-equal key "UNTIL")
           (setf temp (lppop body))
           (setf key (lppop body))
           (setf forms (nconc forms '((when ,temp (loop-finish))))))
          ((string-equal key "WITH")
           (when forms (error "WITH before executable in LOOP BODY."))
           (setf bindings (list* () () bindings))
           (setf key (lppop body))
           (tagbody
             next   (unless (symbolp key) (go set))
                    (when (loop-keyword? key) (go exit))
                    (when (string-equal (string key) "AND")
                      (setf key (lppop body)) (go next))
```

continued

```
            set     (setf temp key)
                    (setf key (lppop body))
                    (cond ((and (symbolp key) (string-equal (string key) "="))
                           (setf key (lppop body))
                           (add-loop-bindings bindings temp key)
                           (setf key (lppop body)))
                          (t (add-loop-bindings bindings temp nil)))
                    (go next)
              exit))
          ((or (setf temp (string-equal key "FOR")) (string-equal key "AS"))
           (setf bindings (list* () () bindings))
           (multiple-value-setq (preset-forms reset-forms body key)
             (loop-for-translate bindings preset-forms reset-forms body
temp)))
          ((or (string-equal key "DO") (string-equal key "DOING"))
           (loop (setf key (pop body))
                 (if (loop-keyword? key) (return nil))
                 (setf forms (nconc forms (list key)))
                 (unless body (return nil))))
          ((member key *loop-collect-keywords* :test #'string-equal)
           (setf temp key)
           (setf bindings (list* () () bindings))
           (let ((exp (lppop body)) (symbol (gensym)))
             (setf key (pop body))
             (when (and key (symbolp key)
                        (member (string key) '("IN" "INTO")
                                :test #'string-equal))
               (setf symbol (lppop body))
               (setf key (pop body)))
             (add-loop-bindings bindings symbol nil)
             (setf forms
                   (nconc forms
                          '((setf ,symbol
                                  ,(loop-collect-form temp symbol exp)))))
             (setf exit-forms
                   (nconc exit-forms (list (list 'return symbol))))))
          )))
```

PUZZLE.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; PUZZLE

(defconstant size 511.)
(defconstant classmax 3.)
(defconstant typemax 12.)

(defconstant true t)
(defconstant false ())
(defvar iii 0)
(defvar kount 0)
(defvar *d* 8.)

(defvar piececount (make-array (1+ classmax) ':initial-element 0))
(defvar class (make-array (1+ typemax) ':initial-element 0))
(defvar piecemax (make-array (1+ typemax) ':initial-element 0))
(defvar puzzle (make-array (1+ size)))
(defvar *p* (make-array (list (1+ typemax) (1+ size))))

(defun fit (i j)
  (let ((end (aref piecemax i)))
    (do ((k 0 (1+ k)))
        ((> k end) true)
      (cond ((aref *p* i k)
             (cond ((aref puzzle (+ j k))
                    (return false)))))))))
```

```
(defun place (i j)
  (let ((end (aref piecemax i)))
    (do ((k 0 (1+ k)))
        ((> k end))
      (cond ((aref *p* i k)
             (setf (aref puzzle (+ j k)) true))))
    (setf (aref piececount (aref class i)) (- (aref piececount (aref class i)) 1))
    (do ((k j (1+ k)))
        ((> k size)

;               (terpri)
;               (princ "Puzzle filled")

        0)
      (cond ((not (aref puzzle k))
                  (return k))))))

(defun puzzle-remove (i j)
  (let ((end (aref piecemax i)))
    (do ((k 0 (1+ k)))
        ((> k end))
      (cond ((aref *p* i k) (setf (aref puzzle (+ j k)) false))))
      (setf (aref piececount (aref class i)) (+ (aref piececount (aref class i)) 1))))

(defun trial (j)
  (let ((k 0))
    (do ((i 0 (1+ i)))
        ((> i typemax) (setq kount (1+ kount)) false)
      (cond ((not (= (aref piececount (aref class i)) 0))
             (cond ((fit i j)
                    (setq k (place i j))
                    (cond ((or (trial k)
                               (= k 0));
                           (format t "%Piece 4D at 4D." (+ i 1) (+ k 1))
                           (setq kount (+ kount 1))
                           (return true))
                          (t (puzzle-remove i j)))))))))

(defun definepiece (iclass ii jj kk)
  (let ((index 0))
    (do ((i 0 (1+ i)))
        ((> i ii))
      (do ((j 0 (1+ j)))
          ((> j jj))
        (do ((k 0 (1+ k)))
            ((> k kk))
          (setq index  (+ i (* *d* (+ j (* *d* k)))))
          (setf (aref *p* iii index) true))))
    (setf (aref class iii) iclass)
    (setf (aref piecemax iii) index)
    (cond ((not (= iii typemax))
           (setq iii (+ iii 1))))))

(defun start ()
  (do ((m 0 (1+ m)))
      ((> m size))
    (setf (aref puzzle m) true))
  (do ((i 1 (1+ i)))
      ((> i 5))
    (do ((j 1 (1+ j)))
        ((> j 5))
      (do ((k 1 (1+ k)))
          ((> k 5))
        (setf (aref puzzle (+ i (* *d* (+ j (* *d* k))))) false))))
  (do ((i 0 (1+ i)))
      ((> i typemax))
    (do ((m 0 (1+ m)))
        ((> m size))
      (setf (aref *p* i m) false)))
  (setq iii 0)
  (definePiece 0 3 1 0)
  (definePiece 0 1 0 3)
  (definePiece 0 0 3 1)
  (definePiece 0 1 3 0)
  (definePiece 0 3 0 1)
  (definePiece 0 0 1 3)
```

continued

```
(definePiece 1 2 0 0)
(definePiece 1 0 2 0)
(definePiece 1 0 0 2)

(definePiece 2 1 1 0)
(definePiece 2 1 0 1)
(definePiece 2 0 1 1)

(definePiece 3 1 1 1)

(setf (aref pieceCount 0) 13.)
(setf (aref pieceCount 1) 3)
(setf (aref pieceCount 2) 1)
(setf (aref pieceCount 3) 1)
(let ((m (+ 1 (* *d* (+ 1 *d*))))
      (n 0)(kount 0))
  (cond ((fit 0 m) (setq n (place 0 m)))
        (t (format t "%Error.")))
  (cond ((trial n)
         (format t "%Success in 4D trials." kount))
        (t (format t "%Failure."))))

(define-timer puzzle "Puzzle" (start))
(qa-attempt "Puzzle" (start) nil)
```

TAK.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; TAK

(defun tak (x y z)
  (declare (type fixnum x y z))
  (if (not (< y x))                                    ;xy
      z
      (tak (tak (1- x) y z)
           (tak (1- y) z x)
           (tak (1- z) x y))))

; TAK with the tail recursion taken out.
(defun trtak (x y z)
  (declare (type fixnum x y z))
  (loop (if (not (< y x))
            (return z)
            (psetq x (tak (1- x) y z)
                   y (tak (1- y) z x)
                   z (tak (1- z) x y)))))

(define-timer tak "Tak" (tak 18. 12. 6.))
(define-timer trtak "Tak with Tail Recusion eliminated" (trtak 18. 12. 6.))

(qa-attempt "Tak" (tak 18. 12. 6.) 7)
(qa-attempt "Tak with Tail Recusion eliminated" (trtak 18. 12. 6.) 7)

; STAK
; TAK using special binding in place of parameter passing.

(defvar *x*)
(defvar *y*)
(defvar *z*)
(proclaim '(type fixnum *x* *y* *z*))

(defun stak (*x* *y* *z*)
  (stak-aux))
```

```
(defun stak-aux ()
  (if (not (< *y* *x*))                          ;xy
      *z*
      (let ((*x* (let ((*x* (1- *x*))
                       (*y* *y*)
                       (*z* *z*))
                   (stak-aux)))
            (*y* (let ((*x* (1- *y*))
                       (*y* *z*)
                       (*z* *x*))
                   (stak-aux)))
            (*z* (let ((*x* (1- *z*))
                       (*y* *x*)
                       (*z* *y*))
                   (stak-aux))))
        (stak-aux))))

(define-timer stak "STak" (stak 18. 12. 6.))
(qa-attempt "STak" (stak 18. 12. 6.) 7)

; CTAK
; TAK using CATCH/THROW.

(defun ctak (x y z)
  (declare (type fixnum x y z))
  (catch 'ctak (ctak-aux x y z)))

(defun ctak-aux (x y z)
  (declare (type fixnum x y z))
  (cond ((not (< y x))                           ;xy
         (throw 'ctak z))
        (t (ctak-aux
             (catch 'ctak
               (ctak-aux (1- x)
                         y
                         z))
             (catch 'ctak
               (ctak-aux (1- y)
                         z
                         x))
             (catch 'ctak
               (ctak-aux (1- z)
                         x
                         y))))))

(define-timer ctak "CTak" (ctak 18. 12. 6.))
(qa-attempt "CTak" (ctak 18. 12. 6.) 7)

; TAKL

(defun listn (n)
   (if (not (= 0 n))
       (cons n (listn (1- n)))))
(defvar 18l )(setq 18l (listn 18))
(defvar 12l )(setq 12l (listn 12))
(defvar 6l  )(setq 6l  (listn 6))

(defun mas (x y z)
  (declare (type list x y z))
  (if (not (shorterp y x))
      z
      (mas (mas (cdr x) y z)
           (mas (cdr y) z x)
           (mas (cdr z) x y))))

(defun shorterp (x y)
  (declare (type list x y))
  (and y (or (null x)
             (shorterp (cdr x)
                       (cdr y)))))

(define-timer takl "TakL" (mas 18l 12l 6l))
(qa-attempt "TakL" (mas 18l 12l 6l) (7 6 5 4 3 2 1))
```

continued

```
; TAKR
; Gross Version to try to trash cache.

(define-timer takr "TakR" (tak0 18. 12. 6.))
(qa-attempt "TakR" (tak0 18. 12. 6.) 7)

(DEFUN TAK0 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK1 (TAK37 (1- X) Y Z)
                   (TAK11 (1- Y) Z X)
                   (TAK17 (1- Z) X Y)))))
(DEFUN TAK1 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK2 (TAK74 (1- X) Y Z)
                   (TAK22 (1- Y) Z X)
                   (TAK34 (1- Z) X Y)))))
(DEFUN TAK2 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK3 (TAK11 (1- X) Y Z)
                   (TAK33 (1- Y) Z X)
                   (TAK51 (1- Z) X Y)))))
(DEFUN TAK3 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK4 (TAK48 (1- X) Y Z)
                   (TAK44 (1- Y) Z X)
                   (TAK68 (1- Z) X Y)))))
(DEFUN TAK4 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK5 (TAK85 (1- X) Y Z)
                   (TAK55 (1- Y) Z X)
                   (TAK85 (1- Z) X Y)))))
(DEFUN TAK5 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK6 (TAK22 (1- X) Y Z)
                   (TAK66 (1- Y) Z X)
                   (TAK2 (1- Z) X Y)))))
(DEFUN TAK6 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK7 (TAK59 (1- X) Y Z)
                   (TAK77 (1- Y) Z X)
                   (TAK19 (1- Z) X Y)))))
(DEFUN TAK7 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK8 (TAK96 (1- X) Y Z)
                   (TAK88 (1- Y) Z X)
                   (TAK36 (1- Z) X Y)))))
(DEFUN TAK8 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK9 (TAK33 (1- X) Y Z)
                   (TAK99 (1- Y) Z X)
                   (TAK53 (1- Z) X Y)))))
(DEFUN TAK9 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK10 (TAK70 (1- X) Y Z)
                    (TAK10 (1- Y) Z X)
                    (TAK70 (1- Z) X Y)))))
(DEFUN TAK10 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK11 (TAK7 (1- X) Y Z)
                    (TAK21 (1- Y) Z X)
                    (TAK87 (1- Z) X Y)))))
```

```
(DEFUN TAK11 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK12 (TAK44 (1- X) Y Z)
                    (TAK32 (1- Y) Z X)
                    (TAK4 (1- Z) X Y)))))
(DEFUN TAK12 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK13 (TAK81 (1- X) Y Z)
                    (TAK43 (1- Y) Z X)
                    (TAK21 (1- Z) X Y)))))
(DEFUN TAK13 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK14 (TAK18 (1- X) Y Z)
                    (TAK54 (1- Y) Z X)
                    (TAK38 (1- Z) X Y)))))
(DEFUN TAK14 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK15 (TAK55 (1- X) Y Z)
                    (TAK65 (1- Y) Z X)
                    (TAK55 (1- Z) X Y)))))
(DEFUN TAK15 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK16 (TAK92 (1- X) Y Z)
                    (TAK76 (1- Y) Z X)
                    (TAK72 (1- Z) X Y)))))
(DEFUN TAK16 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK17 (TAK29 (1- X) Y Z)
                    (TAK87 (1- Y) Z X)
                    (TAK89 (1- Z) X Y)))))
(DEFUN TAK17 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK18 (TAK66 (1- X) Y Z)
                    (TAK98 (1- Y) Z X)
                    (TAK6 (1- Z) X Y)))))
(DEFUN TAK18 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK19 (TAK3 (1- X) Y Z)
                    (TAK9 (1- Y) Z X)
                    (TAK23 (1- Z) X Y)))))
(DEFUN TAK19 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK20 (TAK40 (1- X) Y Z)
                    (TAK20 (1- Y) Z X)
                    (TAK40 (1- Z) X Y)))))
(DEFUN TAK20 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK21 (TAK77 (1- X) Y Z)
                    (TAK31 (1- Y) Z X)
                    (TAK57 (1- Z) X Y)))))
(DEFUN TAK21 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK22 (TAK14 (1- X) Y Z)
                    (TAK42 (1- Y) Z X)
                    (TAK74 (1- Z) X Y)))))
(DEFUN TAK22 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK23 (TAK51 (1- X) Y Z)
                    (TAK53 (1- Y) Z X)
                    (TAK91 (1- Z) X Y)))))
```

continued

```
(DEFUN TAK23 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK24 (TAK88 (1- X) Y Z)
                    (TAK64 (1- Y) Z X)
                    (TAK8 (1- Z) X Y)))))
(DEFUN TAK24 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK25 (TAK25 (1- X) Y Z)
                    (TAK75 (1- Y) Z X)
                    (TAK25 (1- Z) X Y)))))
(DEFUN TAK25 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK26 (TAK62 (1- X) Y Z)
                    (TAK86 (1- Y) Z X)
                    (TAK42 (1- Z) X Y)))))
(DEFUN TAK26 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK27 (TAK99 (1- X) Y Z)
                    (TAK97 (1- Y) Z X)
                    (TAK59 (1- Z) X Y)))))
(DEFUN TAK27 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK28 (TAK36 (1- X) Y Z)
                    (TAK8 (1- Y) Z X)
                    (TAK76 (1- Z) X Y)))))
(DEFUN TAK28 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK29 (TAK73 (1- X) Y Z)
                    (TAK19 (1- Y) Z X)
                    (TAK93 (1- Z) X Y)))))
(DEFUN TAK29 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK30 (TAK10 (1- X) Y Z)
                    (TAK30 (1- Y) Z X)
                    (TAK10 (1- Z) X Y)))))
(DEFUN TAK30 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK31 (TAK47 (1- X) Y Z)
                    (TAK41 (1- Y) Z X)
                    (TAK27 (1- Z) X Y)))))
(DEFUN TAK31 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK32 (TAK84 (1- X) Y Z)
                    (TAK52 (1- Y) Z X)
                    (TAK44 (1- Z) X Y)))))
(DEFUN TAK32 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK33 (TAK21 (1- X) Y Z)
                    (TAK63 (1- Y) Z X)
                    (TAK61 (1- Z) X Y)))))
(DEFUN TAK33 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK34 (TAK58 (1- X) Y Z)
                    (TAK74 (1- Y) Z X)
                    (TAK78 (1- Z) X Y)))))
(DEFUN TAK34 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK35 (TAK95 (1- X) Y Z)
                    (TAK85 (1- Y) Z X)
                    (TAK95 (1- Z) X Y)))))
```

```
(DEFUN TAK35 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK36 (TAK32 (1- X) Y Z)
                    (TAK96 (1- Y) Z X)
                    (TAK12 (1- Z) X Y)))))
(DEFUN TAK36 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK37 (TAK69 (1- X) Y Z)
                    (TAK7 (1- Y) Z X)
                    (TAK29 (1- Z) X Y)))))
(DEFUN TAK37 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK38 (TAK6 (1- X) Y Z)
                    (TAK18 (1- Y) Z X)
                    (TAK46 (1- Z) X Y)))))
(DEFUN TAK38 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK39 (TAK43 (1- X) Y Z)
                    (TAK29 (1- Y) Z X)
                    (TAK63 (1- Z) X Y)))))
(DEFUN TAK39 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK40 (TAK80 (1- X) Y Z)
                    (TAK40 (1- Y) Z X)
                    (TAK80 (1- Z) X Y)))))
(DEFUN TAK40 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK41 (TAK17 (1- X) Y Z)
                    (TAK51 (1- Y) Z X)
                    (TAK97 (1- Z) X Y)))))
(DEFUN TAK41 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK42 (TAK54 (1- X) Y Z)
                    (TAK62 (1- Y) Z X)
                    (TAK14 (1- Z) X Y)))))
(DEFUN TAK42 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK43 (TAK91 (1- X) Y Z)
                    (TAK73 (1- Y) Z X)
                    (TAK31 (1- Z) X Y)))))
(DEFUN TAK43 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK44 (TAK28 (1- X) Y Z)
                    (TAK84 (1- Y) Z X)
                    (TAK48 (1- Z) X Y)))))
(DEFUN TAK44 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK45 (TAK65 (1- X) Y Z)
                    (TAK95 (1- Y) Z X)
                    (TAK65 (1- Z) X Y)))))
(DEFUN TAK45 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK46 (TAK2 (1- X) Y Z)
                    (TAK6 (1- Y) Z X)
                    (TAK82 (1- Z) X Y)))))
(DEFUN TAK46 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK47 (TAK39 (1- X) Y Z)
                    (TAK17 (1- Y) Z X)
                    (TAK99 (1- Z) X Y)))))
```

continued

```
(DEFUN TAK47 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK48 (TAK76 (1- X) Y Z)
                    (TAK28 (1- Y) Z X)
                    (TAK16 (1- Z) X Y)))))
(DEFUN TAK48 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK49 (TAK13 (1- X) Y Z)
                    (TAK39 (1- Y) Z X)
                    (TAK33 (1- Z) X Y)))))
(DEFUN TAK49 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK50 (TAK50 (1- X) Y Z)
                    (TAK50 (1- Y) Z X)
                    (TAK50 (1- Z) X Y)))))
(DEFUN TAK50 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK51 (TAK87 (1- X) Y Z)
                    (TAK61 (1- Y) Z X)
                    (TAK67 (1- Z) X Y)))))
(DEFUN TAK51 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK52 (TAK24 (1- X) Y Z)
                    (TAK72 (1- Y) Z X)
                    (TAK84 (1- Z) X Y)))))
(DEFUN TAK52 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK53 (TAK61 (1- X) Y Z)
                    (TAK83 (1- Y) Z X)
                    (TAK1 (1- Z) X Y)))))
(DEFUN TAK53 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK54 (TAK98 (1- X) Y Z)
                    (TAK94 (1- Y) Z X)
                    (TAK18 (1- Z) X Y)))))
(DEFUN TAK54 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK55 (TAK35 (1- X) Y Z)
                    (TAK5 (1- Y) Z X)
                    (TAK35 (1- Z) X Y)))))
(DEFUN TAK55 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK56 (TAK72 (1- X) Y Z)
                    (TAK16 (1- Y) Z X)
                    (TAK52 (1- Z) X Y)))))
(DEFUN TAK56 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK57 (TAK9 (1- X) Y Z)
                    (TAK27 (1- Y) Z X)
                    (TAK69 (1- Z) X Y)))))
(DEFUN TAK57 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK58 (TAK46 (1- X) Y Z)
                    (TAK38 (1- Y) Z X)
                    (TAK86 (1- Z) X Y)))))
(DEFUN TAK58 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK59 (TAK83 (1- X) Y Z)
                    (TAK49 (1- Y) Z X)
                    (TAK3 (1- Z) X Y)))))
```

```
(DEFUN TAK59 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK60 (TAK20 (1- X) Y Z)
                    (TAK60 (1- Y) Z X)
                    (TAK20 (1- Z) X Y)))))
(DEFUN TAK60 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK61 (TAK57 (1- X) Y Z)
                    (TAK71 (1- Y) Z X)
                    (TAK37 (1- Z) X Y)))))
(DEFUN TAK61 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK62 (TAK94 (1- X) Y Z)
                    (TAK82 (1- Y) Z X)
                    (TAK54 (1- Z) X Y)))))
(DEFUN TAK62 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK63 (TAK31 (1- X) Y Z)
                    (TAK93 (1- Y) Z X)
                    (TAK71 (1- Z) X Y)))))
(DEFUN TAK63 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK64 (TAK68 (1- X) Y Z)
                    (TAK4 (1- Y) Z X)
                    (TAK88 (1- Z) X Y)))))
(DEFUN TAK64 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK65 (TAK5 (1- X) Y Z)
                    (TAK15 (1- Y) Z X)
                    (TAK5 (1- Z) X Y)))))
(DEFUN TAK65 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK66 (TAK42 (1- X) Y Z)
                    (TAK26 (1- Y) Z X)
                    (TAK22 (1- Z) X Y)))))
(DEFUN TAK66 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK67 (TAK79 (1- X) Y Z)
                    (TAK37 (1- Y) Z X)
                    (TAK39 (1- Z) X Y)))))
(DEFUN TAK67 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK68 (TAK16 (1- X) Y Z)
                    (TAK48 (1- Y) Z X)
                    (TAK56 (1- Z) X Y)))))
(DEFUN TAK68 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK69 (TAK53 (1- X) Y Z)
                    (TAK59 (1- Y) Z X)
                    (TAK73 (1- Z) X Y)))))
(DEFUN TAK69 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK70 (TAK90 (1- X) Y Z)
                    (TAK70 (1- Y) Z X)
                    (TAK90 (1- Z) X Y)))))
(DEFUN TAK70 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK71 (TAK27 (1- X) Y Z)
                    (TAK81 (1- Y) Z X)
                    (TAK7 (1- Z) X Y)))))
```

continued

April

```
(DEFUN TAK71 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK72 (TAK64 (1- X) Y Z)
                    (TAK92 (1- Y) Z X)
                    (TAK24 (1- Z) X Y)))))
(DEFUN TAK72 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK73 (TAK1 (1- X) Y Z)
                    (TAK3 (1- Y) Z X)
                    (TAK41 (1- Z) X Y)))))
(DEFUN TAK73 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK74 (TAK38 (1- X) Y Z)
                    (TAK14 (1- Y) Z X)
                    (TAK58 (1- Z) X Y)))))
(DEFUN TAK74 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK75 (TAK75 (1- X) Y Z)
                    (TAK25 (1- Y) Z X)
                    (TAK75 (1- Z) X Y)))))
(DEFUN TAK75 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK76 (TAK12 (1- X) Y Z)
                    (TAK36 (1- Y) Z X)
                    (TAK92 (1- Z) X Y)))))
(DEFUN TAK76 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK77 (TAK49 (1- X) Y Z)
                    (TAK47 (1- Y) Z X)
                    (TAK9 (1- Z) X Y)))))
(DEFUN TAK77 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK78 (TAK86 (1- X) Y Z)
                    (TAK58 (1- Y) Z X)
                    (TAK26 (1- Z) X Y)))))
(DEFUN TAK78 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK79 (TAK23 (1- X) Y Z)
                    (TAK69 (1- Y) Z X)
                    (TAK43 (1- Z) X Y)))))
(DEFUN TAK79 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK80 (TAK60 (1- X) Y Z)
                    (TAK80 (1- Y) Z X)
                    (TAK60 (1- Z) X Y)))))
(DEFUN TAK80 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK81 (TAK97 (1- X) Y Z)
                    (TAK91 (1- Y) Z X)
                    (TAK77 (1- Z) X Y)))))
(DEFUN TAK81 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK82 (TAK34 (1- X) Y Z)
                    (TAK2 (1- Y) Z X)
                    (TAK94 (1- Z) X Y)))))
(DEFUN TAK82 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK83 (TAK71 (1- X) Y Z)
                    (TAK13 (1- Y) Z X)
                    (TAK11 (1- Z) X Y)))))
```

```
(DEFUN TAK83 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK84 (TAK8 (1- X) Y Z)
                    (TAK24 (1- Y) Z X)
                    (TAK28 (1- Z) X Y)))))
(DEFUN TAK84 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK85 (TAK45 (1- X) Y Z)
                    (TAK35 (1- Y) Z X)
                    (TAK45 (1- Z) X Y)))))
(DEFUN TAK85 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK86 (TAK82 (1- X) Y Z)
                    (TAK46 (1- Y) Z X)
                    (TAK62 (1- Z) X Y)))))
(DEFUN TAK86 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK87 (TAK19 (1- X) Y Z)
                    (TAK57 (1- Y) Z X)
                    (TAK79 (1- Z) X Y)))))
(DEFUN TAK87 (X Y Z)  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK88 (TAK56 (1- X) Y Z)
                    (TAK68 (1- Y) Z X)
                    (TAK96 (1- Z) X Y)))))
(DEFUN TAK88 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK89 (TAK93 (1- X) Y Z)
                    (TAK79 (1- Y) Z X)
                    (TAK13 (1- Z) X Y)))))
(DEFUN TAK89 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK90 (TAK30 (1- X) Y Z)
                    (TAK90 (1- Y) Z X)
                    (TAK30 (1- Z) X Y)))))
(DEFUN TAK90 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK91 (TAK67 (1- X) Y Z)
                    (TAK1 (1- Y) Z X)
                    (TAK47 (1- Z) X Y)))))
(DEFUN TAK91 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK92 (TAK4 (1- X) Y Z)
                    (TAK12 (1- Y) Z X)
                    (TAK64 (1- Z) X Y)))))
(DEFUN TAK92 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK93 (TAK41 (1- X) Y Z)
                    (TAK23 (1- Y) Z X)
                    (TAK81 (1- Z) X Y)))))
(DEFUN TAK93 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK94 (TAK78 (1- X) Y Z)
                    (TAK34 (1- Y) Z X)
                    (TAK98 (1- Z) X Y)))))
(DEFUN TAK94 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK95 (TAK15 (1- X) Y Z)
                    (TAK45 (1- Y) Z X)
                    (TAK15 (1- Z) X Y)))))
```

continued

```
(DEFUN TAK95 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK96 (TAK52 (1- X) Y Z)
                    (TAK56 (1- Y) Z X)
                    (TAK32 (1- Z) X Y)))))
(DEFUN TAK96 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK97 (TAK89 (1- X) Y Z)
                    (TAK67 (1- Y) Z X)
                    (TAK49 (1- Z) X Y)))))
(DEFUN TAK97 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK98 (TAK26 (1- X) Y Z)
                    (TAK78 (1- Y) Z X)
                    (TAK66 (1- Z) X Y)))))
(DEFUN TAK98 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK99 (TAK63 (1- X) Y Z)
                    (TAK89 (1- Y) Z X)
                    (TAK83 (1- Z) X Y)))))
(DEFUN TAK99 (X Y Z)
  (declare (type fixnum x y z))
    (COND ((NOT (< Y X)) Z)
          (T (TAK0 (TAK0 (1- X) Y Z)
                   (TAK0 (1- Y) Z X)
                   (TAK0 (1- Z) X Y)))))
```

TIMING.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
;  timimg routines

(defconstant internal-time-units-per-second 100)

(defun get-internal-run-time ()
  (multiple-value-bind (ignore1 ignore2 ignore3 cx dx)
      (sys:%sysint #x21 #x2c00 0 0 0)
    (+ (* (lsh cx -8)      60 60 100)
       (* (logand cx #xFF)    60 100)
       (* (lsh dx -8)            100)
       (logand dx #xFF))))

(defun timed-duration (fn)
  (let ((start-run (get-internal-run-time)))
    (funcall fn)
    (let ((end-run (get-internal-run-time)))
      (float (/ (- end-run start-run) internal-time-units-per-second)))))

(defparameter *minimum-tests* 1)
(defparameter *minimum-duration* 10.0)

(defun multiple-timed-duration (fn)
  (let* ((total-run-time (timed-duration fn))
  (repeats (max *minimum-tests*
          (ceiling *minimum-duration*
    (if (zerop total-run-time) 1        total-run-time)))))
    (do ((count repeats (- count 1)))
 ((< count 2) (values total-run-time repeats))
      (incf total-run-time (timed-duration fn)))))

(defvar *all-timers* nil)
(defvar *bad-timers* '(tak boyer))
```

```
(defmacro define-timer (name documentation &body body)
  `(progn (pushnew ',name *all-timers*)
   (setf (get ',name 'timing-function)
  ,(if (and (= (length body) 1) (= (length (first body)) 1))
       (list 'quote (first (first body)))
       `#'(lambda () . ,body)))
   (setf (get ',name 'timing-documentation) ,documentation)))

(defun run-tests (&optional file)
  (if (null file) (run-tests1 't)
      (with-open-file (stream file :direction :output) (run-tests1 stream))))

(defun run-tests1 (stream)
  (describe-implementation stream)
  (do ((tests *all-timers* (cdr tests))) ((null tests) '*)
    (cond ((member (first tests) *bad-timers*)
    (format stream "&Run of A punted due to stack group reset.%"
     (get (first tests) 'timing-documentation)))
   (t (sys::gc)
      (multiple-value-bind (answer error?)
          (ignore-errors (run-one (first tests) stream))
        (if error? (format stream "%    ERROR: A%" error?)))))))

(defun run-one (name &optional (stream *terminal-io*))
  (unless (get name 'timing-documentation)
    (error "&There's no such benchmark as S.%" name))
  (format stream "&Running A . . ." (get name 'timing-documentation))
  (multiple-value-bind (time n-runs)
      (multiple-timed-duration (get name 'timing-function))
    (format stream "%    time: D seconds (based on D call"
     (/ time n-runs) n-runs)
    (unless (= n-runs 1) (write-char #\s stream))
    (format stream ")%" time n-runs)))

(defun describe-implementation (&optional (stream *standard-output*))
  (format stream "&Lisp Type:     A" (lisp-implementation-type))
  (format stream "&Lisp Version:  A" (lisp-implementation-version))
  #+:Large-Memory
  (format stream "&Machine Type:  IBM-PC/AT")
  #-:Large-Memory
  (format stream "&Machine Type:  IBM-PC/XT")
  (format stream "&Features:      A" (car *features*))
  (if (cdr *features*) (format stream ", "))
  (do ((features (cdr *features*) (cdr features))
       (offset (+ 17 (length (string (car *features*))))))
      ((null features))
    (let* ((feature (string (car features))) (lth (length feature)))
      (cond ((> (setq offset (+ offset 2 lth))76)
      (setq offset (+ 15 lth))
      (format stream "&               A" feature))
     (t (format stream "A" feature)))
      (when (cdr features)
  (setq offset (+ offset 2))
  (format stream ", "))))
   (format stream "%%"))


 (defvar *benchmark-files*
  '("DESTRUCT"
    "IO"
    "FRPOLY"
    "TRIANG"
    ;"PUZZLE"
    ;"FFT"
    "DIV"
    "DERIV"
    "TRAVERSE"
    "BROWSE"
    "BOYER"
    "TAK"
    ))


 (defmacro qa-attempt (&body stuff) (list 'quote stuff))
```

continued

```
(defun benchmark-file (file) (merge-pathnames "C:>GCLISP2>" file))

(defun load-benchmark-files ()
  (mapc #'(lambda (file) (load (benchmark-file file))) *benchmark-files*))

(defun compile-benchmark-files (&optional load?)
  (mapc #'(lambda (file) (compile-file (benchmark-file file) :load load?))
        *benchmark-files*))
```

TRAVERSE.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; TRAVERSE
; Benchmark to create once and traverse a Structure

(defstruct node
  (parents ())
  (sons ())
  (sn (snb))
  (entry1 ())
  (entry2 ())
  (entry3 ())
  (entry4 ())
  (entry5 ())
  (entry6 ())
  (mark ()))

(defvar sn 0)
(defvar rand 21.)
(defvar count 0)
(defvar marker nil)
(defvar root)

(defun snb () (setq sn (1+ sn)))

(defun seed () (setq rand 21.))

(defun traverse-random () (setq rand (mod (* rand 17.) 251.)))

(defun traverse-remove (n q)
       (cond ((eq (cdr (car q)) (car q))
              (prog2 () (caar q) (rplaca q ())))
             ((= n 0)
              (prog2 () (caar q)
                     (do ((p (car q) (cdr p)))
                         ((eq (cdr p) (car q))
                          (rplaca q
                                  (rplacd p (cdr (car q))))))))
             (t (do ((n n (1- n))
                     (q (car q) (cdr q))
                     (p (cdr (car q)) (cdr p)))
                    ((= n 0) (prog2 () (car q) (rplacd q p)))))))

(defun traverse-select (n q)
       (do ((n n (1- n))
                  (q (car q) (cdr q)))
           ((= n 0) (car q))))

(defun add (a q)
       (cond ((null q)
              '(,(let ((x '(,a)))
                   (rplacd x x) x)))
             ((null (car q))
              (let ((x '(,a)))
                   (rplacd x x)
                   (rplaca q x)))
             (t (rplaca q
                        (rplacd (car q) '(,a ,.(cdr (car q))))))))
```

```
(defun create-structure (n)
       (let ((a '(,(make-node))))
            (do ((m (1- n) (1- m))
                 (p a))
                ((= m 0) (setq a '(,(rplacd p a)))
                         (do ((unused a)
                              (used (add (traverse-remove 0 a) ()))
                              (x) (y))
                             ((null (car unused))
                              (find-root (traverse-select 0 used) n))
                             (setq x (traverse-remove (rem (traverse-random)n) unused))
                             (setq y (traverse-select (rem (traverse-random)n) used))
                             (add x used)
                             (setf (node-sons y) '(,x .,(node-sons y)))
                             (setf (node-parents x) '(,y .,(node-parents x)))))
                (push (make-node) a))))

(defun find-root (node n)
 (do ((n n (1- n)))
     ((= n 0) node)
     (cond ((null (node-parents node))
            (return node))
           (t (setq node (car (node-parents node)))))))

(defun travers (node mark)
       (cond ((eq (node-mark node) mark) ())
             (t (setf (node-mark node) mark)
                (setq count (1+ count))
                (setf (node-entry1 node) (not (node-entry1 node)))
                (setf (node-entry2 node) (not (node-entry2 node)))
                (setf (node-entry3 node) (not (node-entry3 node)))
                (setf (node-entry4 node) (not (node-entry4 node)))
                (setf (node-entry5 node) (not (node-entry5 node)))
                (setf (node-entry6 node) (not (node-entry6 node)))
                (do ((sons (node-sons node) (cdr sons)))
                    ((null sons) ())
                    (travers (car sons) mark))))))

(defun traverse (root)
       (let ((count 0))
            (travers root (setq marker (not marker)))
            count))

(qa-attempt "Traverse init" (setq root (create-structure 100.)) nil)

(qa-attempt "Traverse"
   (do ((i 50. (1- i)))
       ((= i 0))
    (traverse root)
    (traverse root)
    (traverse root)
    (traverse root)
    (traverse root))
  nil)

(define-timer traverse "Traverse, Traverse"
  (do ((i 50. (1- i)))
      ((= i 0))
    (traverse root)
    (traverse root)
    (traverse root)
    (traverse root)    (traverse root)))

(define-timer traverse-init "Traverse, Initialize"
  (prog2 (setq root (create-structure 100.)) ()))
```

continued

TRIANG.LSP Contributed by: Ernest R. Tello
TEXT "The GCLISP 286 Developer," by Ernest R. Tello. April 1987, page 242.

```
; TRIANG

(defvar board '#(1 1 1 1 1 0 1 1 1 1 1 1 1 1 1 1))
(defvar sequence (make-array 14. ':initial-element 0.))
(defvar *a* '#(1 2 4 3 5 6 1 3 6 2 5 4 11. 12. 13. 7 8. 4
                 4 7 11. 8. 12. 13. 6 10. 15. 9. 14. 13. 13. 14. 15. 9. 10. 66))
(defvar *b* '#(2 4 7 5 8. 9. 3 6 10. 5 9. 8. 12. 13. 14. 8. 9. 5
                 2 4 7 5 8. 9. 3 6 10. 5 9. 8. 12. 13. 14. 8. 9. 5 5))
(defvar *c* '#(4 7 11. 8. 12. 13. 6 10. 15. 9. 14. 13. 13. 14. 15. 9. 10. 6
                 1 2 4 3 5 6 1 3 6 2 5 4 11. 12. 13. 7 8. 4 4))
(defvar answer)
(defvar final)

(defun last-position ()
  (do ((i 1 (1+ i)))
      ((= i 16.) 0)
    (if (= 1 (aref board i))
        (return i))))

(defun try (i depth)
       (cond ((= depth 14)
              (let ((lp (last-position)))
                   (unless (member lp final)
                     (push lp final)))
              (push (cdr (coerce sequence 'list)) answer) t)
             ((and (= 1 (aref board (aref *a* i)))
                   (= 1 (aref board (aref *b* i)))
                   (= 0 (aref board (aref *c* i))))
              (setf (aref board (aref *a* i)) 0)
              (setf (aref board (aref *b* i)) 0)
              (setf (aref board (aref *c* i)) 1)
              (setf (aref sequence depth) i)
              (do ((j 0 (1+ j))
                   (depth (1+ depth)))
                  ((or (= j 36.)
                       (try j depth)) ()))
              (setf (aref board (aref *a* i)) 1)
              (setf (aref board (aref *b* i)) 1)
              (setf (aref board (aref *c* i)) 0) ())))

(defun gogogo (i)
  (dotimes (j 16)
    (setf (aref board j) 1))
  (setf (aref board 5) 0)
  (let ((answer ())
        (final ()))
    (try i 1)))

(define-timer triang "Triang" (gogogo 22.))
(qa-attempt "Triang" (gogogo 22.) nil)

(defun triang-test ()
  (dotimes (j 16)
    (setf (aref board j) 1))
  (setf (aref board 5) 0)
  (let ((answer ())
        (final ()))
    (try 22. 1)
    (= (length answer) 775.)))
```

May

README.1ST Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
Documentation

README   1ST      860   3-08-87  11:49a
ADVSYS   DOC    30536   7-20-86  12:15p

Sample adventure source

OSAMPLE  ADV     4355   7-20-86   2:15p
OBJECTS  ADI    13193   7-20-86   2:16p

Definitions (used by ADVCOM and ADVINT)

ADVDBS   H       5584   7-20-86  12:48p

Compiler source code (ADVCOM)

ADVCOM   H       1399   7-20-86  12:48p
ADVAVL   H        782   7-20-86  12:49p
ADVCOM   C      15476   7-20-86  12:40p
ADVFCN   C      14285   7-20-86  12:41p
ADVSCN   C       6362   7-20-86  12:41p
ADVEXP   C      10829   7-20-86  12:42p
ADVAVL   C       3750   7-20-86  12:42p
ADVFIO   C        998   7-20-86  12:42p

Interpreter source code (ADVINT)

ADVINT   H        383   7-20-86  12:50p
ADVMSG   C       2682   7-20-86  12:43p
ADVTRM   C       3243   7-20-86  12:43p
ADVPRS   C       7269   7-20-86  12:43p
ADVDBS   C      11051   7-20-86  12:44p
ADVINT   C       2569   7-20-86  12:44p
ADVJUNK  C       1848   7-19-86   7:24p
ADVEXE   C       6202   7-20-86  12:44p
```

ADVCOM.H Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advcom.h - adventure compiler definitions */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/
#include <stdio.h>
#include <ctype.h>

/* limits */
#define TKNSIZE         50      /* maximum token size */
#define OSIZE           104     /* maximum object size (O_SIZE/2 + OPMAX*2) */
#define OPMAX           50      /* maximum # properties/object */
#define WMAX            500     /* maximum number of words */
#define OMAX            500     /* maximum number of objects */
#define AMAX            200     /* maximum number of actions */
```

continued

```
#define DMAX            16384   /* maximum data space */
#define CMAX            16384   /* maximum code space */
#define FMAX            20      /* file name maximum */

/* useful definitions */
#define TRUE            1
#define FALSE           0
#define EOS             '\0'

/* token definitions */
#define T_OPEN          1
#define T_CLOSE         2
#define T_STRING        3
#define T_IDENTIFIER    4
#define T_NUMBER        5
#define T_EOF           6

/* symbol types */
#define ST_OBJECT       1
#define ST_ACTION       2
#define ST_VARIABLE     3
#define ST_CONSTANT     4
#define ST_PROPERTY     5

/* symbol structure */
typedef struct symbol {
    char *s_name;               /* symbol name */
    int s_type;                 /* symbol type */
    int s_value;                /* symbol value */
    struct symbol *s_next;      /* next symbol in table */
} SYMBOL;

/* function argument structure */
typedef struct argument {
    char *arg_name;             /* argument name */
    struct argument *arg_next;  /* next argument */
} ARGUMENT;
```

ADVAVL.H Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advavl.h - avl tree definitions */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

typedef struct tree {
    struct tnode *tr_root;      /* root node */
    int tr_cnt;                 /* count of entries */
} TREE;

typedef struct tnode {
    int tn_b;                   /* balance flag */
    struct tnode *tn_llink;     /* left subtree */
    struct tnode *tn_rlink;     /* right subtree */
    char *tn_key;               /* word */
    int tn_word;                /* word number */
} TNODE;

#define LLINK(n)        ((n)->tn_llink)
#define RLINK(n)        ((n)->tn_rlink)
#define KEY(n)          ((n)->tn_key)
#define WORD(n)         ((n)->tn_word)
#define B(n)            ((n)->tn_b)
#define tentries(t)     ((t)->tr_cnt)
```

ADVDBS.H Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advdbs.h - adventure database definitions */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

/* useful constants */
#define T       -1
#define NIL     0
#define WRDSIZE 6

/* data structure version number */
#define VERSION         102
/* file header offsets */
#define HDR_LENGTH      0       /* length of header in bytes */
#define HDR_MAGIC       2       /* magic information (6 bytes) */
#define HDR_VERSION     8       /* data structure version number */
#define HDR_ANAME       10      /* adventure name (18 bytes) */
#define HDR_AVERSION    28      /* adventure version number */
#define HDR_WTABLE      30      /* offset to word table */
#define HDR_WTYPES      32      /* offset to word type table */
#define HDR_OTABLE      34      /* offset to object table */
#define HDR_ATABLE      36      /* offset to action table */
#define HDR_VTABLE      38      /* offset to variable table */
#define HDR_DBASE       40      /* offset to base of data space */
#define HDR_CBASE       42      /* offset to base of code space */
#define HDR_DATBLK      44      /* first data block */
#define HDR_MSGBLK      46      /* first message text block */
#define HDR_INIT        48      /* initialization code */
#define HDR_UPDATE      50      /* update code */
#define HDR_BEFORE      52      /* code to execute before verb handler */
#define HDR_AFTER       54      /* code to execute after verb handler */
#define HDR_ERROR       56      /* error handler code */
#define HDR_SAVE        58      /* save area offset */
#define HDR_SLEN        60      /* save area length */
#define HDR_SIZE        62      /* size of header */

/* word types */
#define WT_UNKNOWN      0
#define WT_VERB         1
#define WT_NOUN         2
#define WT_ADJECTIVE    3
#define WT_PREPOSITION  4
#define WT_CONJUNCTION  5
#define WT_ARTICLE      6

/* object fields */
#define O_CLASS         0
#define O_NOUNS         2
#define O_ADJECTIVES    4
#define O_NPROPERTIES   6
#define O_PROPERTIES    8
#define O_SIZE          8

/* action fields */
#define A_VERBS         0
#define A_PREPOSITIONS  2
#define A_FLAG          4
#define A_MASK          5
#define A_CODE          6
#define A_SIZE          8

/* link fields */
#define L_DATA          0
#define L_NEXT          2
#define L_SIZE          4

/* property flags */
#define P_CLASS         0x8000  /* class property */
```

continued

```
/* action flags */
#define A_ACTOR          0x01    /* actor */
#define A_DOBJECT        0x02    /* direct object */
#define A_IOBJECT        0x04    /* indirect object */

/* opcodes */#define OP_BRT          0x01    /* branch on true */
#define OP_BRF           0x02    /* branch on false */
#define OP_BR            0x03    /* branch unconditionally */
#define OP_T             0x04    /* load top of stack with t */
#define OP_NIL           0x05    /* load top of stack with nil */
#define OP_PUSH          0x06    /* push nil onto stack */
#define OP_NOT           0x07    /* logical negate top of stack */
#define OP_ADD           0x08    /* add two numeric expressions */
#define OP_SUB           0x09    /* subtract two numeric expressions */
#define OP_MUL           0x0A    /* multiply two numeric expressions */
#define OP_DIV           0x0B    /* divide two numeric expressions */
#define OP_REM           0x0C    /* remainder of two numeric expressions */
#define OP_BAND          0x0D    /* bitwise and of two numeric expressions */
#define OP_BOR           0x0E    /* bitwise or of two numeric expressions */
#define OP_BNOT          0x0F    /* bitwise not of two numeric expressions */
#define OP_LT            0x10    /* less than */
#define OP_EQ            0x11    /* equal to */
#define OP_GT            0x12    /* greater than */
#define OP_LIT           0x13    /* load literal */
#define OP_VAR           0x14    /* load a variable value */
#define OP_GETP          0x15    /* get the value of an object property */
#define OP_SETP          0x16    /* set the value of an object property */
#define OP_SET           0x17    /* set the value of a variable */
#define OP_PRINT         0x18    /* print messages */
#define OP_TERPRI        0x19    /* terminate the print line */
#define OP_PNUMBER       0x1A    /* print a number */
#define OP_FINISH        0x1B    /* finish handling this command */
#define OP_CHAIN         0x1C    /* chain to the next handler */
#define OP_ABORT         0x1D    /* abort this command */
#define OP_EXIT          0x1E    /* exit the program */
#define OP_RETURN        0x1F    /* return from interpreter */
#define OP_CALL          0x20    /* call a function */
#define OP_SVAR          0x21    /* short load a variable */
#define OP_SSET          0x22    /* short set a variable */
#define OP_SPLIT         0x23    /* short load a positive literal */
#define OP_SNLIT         0x24    /* short load a negative literal */
#define OP_YORN          0x25    /* yes-or-no predicate */
#define OP_SAVE          0x26    /* save data structures */
#define OP_RESTORE       0x27    /* restore data structures */
#define OP_ARG           0x28    /* load an argument value */
#define OP_ASET          0x29    /* set an argument value */
#define OP_TMP           0x2A    /* load a temporary variable value */
#define OP_TSET          0x2B    /* set a temporary variable */
#define OP_TSPACE        0x2C    /* allocate temporary variable space */
#define OP_CLASS         0x2D    /* get the class of an object */
#define OP_MATCH         0x2E    /* match a noun phrase with an object */
#define OP_PNOUN         0x2F    /* print a noun phrase */
#define OP_RESTART       0x30    /* restart the current game */
#define OP_RAND          0x31    /* generate a random number */
#define OP_RNDMIZE       0x32    /* seed the random number generator */
#define OP_SEND          0x33    /* send a message to an object */

#define OP_XVAR          0x40    /* extra short load a variable */
#define OP_XSET          0x60    /* extra short set a variable */
#define OP_XPLIT         0x80    /* extra short load a positive literal */
#define OP_XNLIT         0xC0    /* extra short load a negative literal */

/* builtin variables */
#define V_ACTOR           1      /* actor noun phrase number */
#define V_ACTION          2      /* action from parse */
#define V_DOBJECT         3      /* first direct object noun phrase number */
#define V_NDOBJECTS       4      /* number of direct object noun phrases */
#define V_IOBJECT         5      /* indirect object noun phrase number */
#define V_OCOUNT          6      /* total object count */
```

ADVCOM.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advcom.c - a compiler for adventure games */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advcom.h"
#include "advavl.h"
#include "advdbs.h"

/* symbol tables */
SYMBOL *symbols;
ARGUMENT *arguments;
ARGUMENT *temporaries;

/* adventure id information */
char aname[19];
int aversion;

/* word table */
int wtable[WMAX+1],wcnt;

/* object table */
int otable[OMAX+1],ocnt;

/* action table */
int atable[AMAX+1],acnt;

/* constant, variable and property symbol counts */
int ccnt,vcnt,pcnt;

/* data and code space */
char *data,*code;
int dptr,cptr;

/* buffer for building an object */
int objbuf[OSIZE];
int nprops;
/* global variables */
char ifile[FMAX];       /* input file name */
char ofile[FMAX];       /* output file name */
FILE *ifp;              /* input file pointer */
unsigned int msgoff;    /* message section offset */
TREE *words;            /* word tree */
int curwrd;             /* current word number */
int curobj;             /* current object */
int curact;             /* current action */
int def_flag;           /* default action flag value */
int def_mask;           /* default action mask value */

/* header information variables */
int h_init;             /* initialization code */
int h_update;           /* update code */
int h_before;           /* before handler code */
int h_after;            /* after handler code */
int h_error;            /* error handling code */

/* external routines */
extern char *malloc();
extern char *calloc();
extern TREE *tnew();

/* external variables */
extern int errcount;    /* error count */
extern int t_value;     /* token value */
extern char t_token[];  /* token string */
extern char *t_names[]; /* token names */
extern long ad_foff;    /* data file offset */
```

continued

```
/* forward declarations */
SYMBOL *sfind();
SYMBOL *senter();
char *save();

/* main - the main routine */
main(argc,argv)
  int argc; char *argv[];
{
    int tkn,obj,i;

    /* initialize */
#ifdef MAC
    macinit(ifile,ofile);
#else
    printf("ADVCOM v1.2 - Copyright (c) 1986, by David Betz\n");
#endif
    wcnt = ocnt = acnt = ccnt = vcnt = pcnt = msgoff = 0;
    symbols = NULL; arguments = temporaries = NULL;
    h_init = h_update = h_before = h_after = h_error = NIL;
    def_flag = def_mask = 0;
    aname[0] = 0;
    sinit();
    /* setup the code and data space */
    if ((data = calloc(1,DMAX)) == 0)
        fail("insufficient memory");
    if ((code = calloc(1,CMAX)) == 0)
        fail("insufficient memory");
    dptr = cptr = 1;    /* make sure nothing has a zero offset */

    /* get the file name */
#ifndef MAC
    if (argc < 2)
        fail("usage: advcom <file> [ <ofile> ]");
    strcpy(ifile,argv[1]); strcat(ifile,".adv");
    strcpy(ofile,(argc < 3 ? argv[1] : argv[2])); strcat(ofile,".dat");
#endif

    /* open the input file */
    if ((ifp = fopen(ifile,"r")) == NULL)
        fail("can't open input file");

    /* create and initialize the output file */
    ad_create(ofile);
    for (i = 0; i++ < 512; ad_putc('\0'))
        ;

    /* create the word tree */
    words = tnew();

    /* enter builtin constants */
    center("t",-1);
    center("nil",0);

    /* enter the builtin variables */
    venter("$actor");
    venter("$action");
    venter("$dobject");
    venter("$ndobjects");
    venter("$iobject");
    venter("$ocount");

    /* enter the preposition "to" */
    add_word("to",WT_PREPOSITION);

    /* process statements until end of file */
    while ((tkn = token()) == T_OPEN) {
        frequire(T_IDENTIFIER);

        /* identification statement */
        if (match("adventure"))
            do_adventure();

        /* vocabulary statements */
        else if (match("adjective"))
            do_word(WT_ADJECTIVE);
```

```
    else if (match("preposition"))
        do_word(WT_PREPOSITION);
    else if (match("conjunction"))
        do_word(WT_CONJUNCTION);
    else if (match("article"))
        do_word(WT_ARTICLE);
    else if (match("synonym"))
        do_synonym();

    /* constant, variable, function and default definition statements */
    else if (match("define"))
        do_define();
    else if (match("variable"))
        do_variable();
    else if (match("default"))
        do_default();

    /* property definition statement */
    else if (match("property"))
        do_defproperty();

    /* handle the init, before and after code statements */
    else if (match("init"))
        h_init = do_code(t_token);
    else if (match("update"))
        h_update = do_code(t_token);
    else if (match("before"))
        h_before = do_code(t_token);
    else if (match("after"))
        h_after = do_code(t_token);
    else if (match("error"))
        h_error = do_code(t_token);

    /* action definition statement */
    else if (match("action"))
        do_action();

    /* object definition statements */
    else if (match("object"))
        do_object(t_token,NIL);

    /* object instance definition statements */
    else if (obj = ofind(t_token))
        do_object(t_token,obj);

    /* error, unknown statement */
    else
        error("Unknown statement type");
}
require(tkn,T_EOF);

    /* close the input file */
    fclose(ifp);

    /* output the data structures */
    output();
    /* close the output file */
    ad_close();
}

/* getvalue - get a value */
int getvalue()
{
    SYMBOL *sym;

    switch (token()) {
    case T_IDENTIFIER:  if (sym = sfind(t_token))
                            return (sym->s_value);
                        return (oenter(t_token));
    case T_NUMBER:      return (t_value);
    case T_STRING:      return (t_value);
    default:            error("Expecting identifier, number or string");
                        return (0);
    }
}
```

continued

```
/* dalloc - allocate data space */
int dalloc(size)
  int size;
{
    if ((dptr += size) > DMAX)
        fail("out of data space");
    return (dptr - size);
}

/* add_word - add a word to the dictionary */
int add_word(str,type)
  char *str; int type;
{
    if ((curwrd = tfind(words,str)) == NIL) {
        if (wcnt < WMAX) {
            curwrd = ++wcnt;
            wtable[curwrd] = type;
            tenter(words,str);
        }
        else {
            error("too many words");
            curwrd = 0;
        }
    }
    else if (wtable[curwrd] == WT_UNKNOWN)
        wtable[curwrd] = type;
    else if (type != WT_UNKNOWN && type != wtable[curwrd])
        error("Ambiguous word type");
    return (curwrd);
}

/* add_synonym - add a synonym to a word */
int add_synonym(str,wrd)
  char *str; int wrd;{
    curwrd = wrd;
    return (tenter(words,str));
}

/* getword - get a word from an object field */
int getword(off)
  int off;
{
    return ((data[off] & 0xFF) | (data[off+1] << 8));
}

/* putword - put a word into an object field */
putword(off,dat)
  int off,dat;
{
    data[off] = dat;
    data[off+1] = dat >> 8;
}

/* getbyte - get a byte from an object field */
int getbyte(off)
  int off;
{
    return (data[off]);
}

/* putbyte - put a byte into an object field */
putbyte(off,dat)
  int off,dat;
{
    data[off] = dat;
}

/* output - output the binary data structures */
output()
{
    int woff,wsize;     /* word table offset and size */
    int ooff,osize;     /* object table offset and size */
    int aoff,asize;     /* action table offset and size */
    int toff,tsize;     /* word type table offset and size */
    int voff,vsize;     /* variable table offset and size */
    int soff,ssize;     /* save area offset and size */
```

```
int dsize;          /* data size without dictionary */
int dbase,cbase,size,mblk,dblk,i;

/* make sure the adventure id information is present */
if (aname[0] == 0) {
    xerror("no adventure identification information");
    strcpy(aname,"ADVENTURE");
    aversion = 0;
}

/* pad the remainder of this message block */
while (msgoff & 0x007F)        { ad_putc('\0'); ad_putc('\0');
                                 ad_putc('\0'); ad_putc('\0');msgoff++; }

/* save the size of the data area before the dictionary */
dsize = dptr;

/* insert the vocabulary into the data array */
woutput(words->tr_root);

/* compute table offsets */
woff = 0;           wsize = tentries(words) * 2 + 2;
toff = woff + wsize; tsize = wcnt;
ooff = toff + tsize; osize = ocnt * 2 + 2;
aoff = ooff + osize; asize = acnt * 2 + 2;
voff = aoff + asize; vsize = vcnt * 2 + 2;
dbase = voff + vsize;
cbase = dbase + dptr;

/* compute the resident structure size */
size = wsize+tsize+osize+asize+vsize+dptr+cptr;

/* set the save area parameters */
soff = voff; ssize = vsize + dsize;

/* compute the first block for message text */
mblk = 1;
dblk = (int)(ad_foff >> 9);

/* output the word table */
word_out(tentries(words));
wtoutput(words->tr_root);

/* output the word type table */
for (i = 1; i <= wcnt; i++)
    byte_out(wtable[i]);

/* output the object table */
word_out(ocnt);
for (i = 1; i <= ocnt; i++) {
    if (otable[i] == NIL)
        undef_object(i);
    word_out(otable[i]);
}

/* output the action table */
word_out(acnt);
for (i = 1; i <= acnt; i++)
    word_out(atable[i]);

/* beginning of saveable data */

/* output the variable table */
word_out(vcnt);
for (i = 1; i <= vcnt; i++)
    word_out(NIL);
/* output the data space */
for (i = 0; i < dptr; )
    byte_out(data[i++]);

/* end of saveable data */

/* output the code space */
for (i = 0; i < cptr; )
    byte_out(code[i++]);
```

continued

```
    /* output the file header */
    ad_seek(0L);
    word_out(size);      /* resident structure size */
    str_out("ADVSYS",6);/* magic information */
    word_out(VERSION);  /* data structure version number */
    str_out(aname,18);  /* adventure name */
    word_out(aversion); /* adventure version number */
    word_out(woff);     /* word table offset */
    word_out(toff);     /* word type table offset */
    word_out(ooff);     /* object table offset */
    word_out(aoff);     /* action table offset */
    word_out(voff);     /* variable table offset */
    word_out(dbase);    /* base of data */
    word_out(cbase);    /* base of code */
    word_out(dblk);     /* first data block */
    word_out(mblk);     /* first message text block */
    word_out(h_init);   /* initialization code */
    word_out(h_update); /* update code */
    word_out(h_before); /* before handler code */
    word_out(h_after);  /* after handler code */
    word_out(h_error);  /* error handling code */
    word_out(soff);     /* save area offset */
    word_out(ssize);    /* save area size */

    /* show statistics */
    printf("[ words:      %d ]\n",tentries(words));
    printf("[ word types: %d ]\n",wcnt);
    printf("[ objects:    %d ]\n",ocnt);
    printf("[ actions:    %d ]\n",acnt);
    printf("[ variables:  %d ]\n",vcnt);
    printf("[ data:       %d ]\n",dsize);
    printf("[ code:       %d ]\n",cptr);
    printf("[ dictionary: %d ]\n",dptr-dsize);
    printf("[ text:       %ld ]\n",(long) msgoff * 4L);
    printf("[ save area:  %d ]\n",ssize);
    printf("[ errors:     %d ]\n",errcount);
#ifdef MAC
    macpause();
#endif
}

/* woutput - output the word data */
woutput(node)
  TNODE *node;{
    int wnum,wrd;

    if (node) {
        woutput(LLINK(node));
        wnum = WORD(node);
        wrd = WORD(node) = dalloc(strlen(KEY(node))+3);
        putword(wrd,wnum);
        strcpy(data+wrd+2,KEY(node));
        if (wtable[wnum] == WT_UNKNOWN)
            printf("Type of word %s is unknown\n",KEY(node));
        woutput(RLINK(node));
    }
}

/* wtoutput - output the word table */
wtoutput(node)
  TNODE *node;
{
    if (node) {
        wtoutput(LLINK(node));
        word_out(WORD(node));
        wtoutput(RLINK(node));
    }
}

/* undef_object - complain about an undefined object */
undef_object(n)
  int n;
{
    char msg[100];
    SYMBOL *sym;
```

```
    for (sym = symbols; sym != NULL; sym = sym->s_next)
        if (sym->s_type == ST_OBJECT && n == sym->s_value) {
            sprintf(msg,"Object %s is undefined",sym->s_name);
            xerror(msg);
            break;
        }
}

/* str_out - output a string */
str_out(str,len)
  char *str; int len;
{
    while (len--)
        byte_out(*str++);
}

/* word_out - output a word */
word_out(dat)
  int dat;
{
    byte_out(dat);
    byte_out(dat >> 8);}

/* byte_out - output a byte */
byte_out(dat)
  int dat;
{
    ad_putc((dat - 30) & 0xFF);
}

/* oenter - enter an object into the symbol table */
int oenter(name)
  char *name;
{
    SYMBOL *sym;

    if (sym = sfind(name)) {
        if (sym->s_type != ST_OBJECT)
            error("Not an object");
        return (sym->s_value);
    }
    if (ocnt < OMAX) {
        senter(name,ST_OBJECT,++ocnt);
        otable[ocnt] = NIL;
    }
    else
        error("too many objects");
    return (ocnt);
}

/* ofind - find an object in the symbol table */
int ofind(name)
  char *name;
{
    SYMBOL *sym;

    if (sym = sfind(name)) {
        if (sym->s_type != ST_OBJECT)
            return (NIL);
        return (sym->s_value);
    }
    return (NIL);
}

/* aenter - enter an action into the symbol table */
int aenter(name)
  char *name;
{
    SYMBOL *sym;

    if (sym = sfind(name)) {
        if (sym->s_type != ST_ACTION)
            error("Not an action");
        return (sym->s_value);
```

continued

```
    }
    if (acnt < AMAX) {          senter(name,ST_ACTION,++acnt);
        atable[acnt] = NIL;
    }
    else
        error("too many actions");
        return (acnt);
    }

    /* venter - enter a variable into the symbol table */
    int venter(name)
      char *name;
    {
        SYMBOL *sym;

        if (sym = sfind(name)) {
            if (sym->s_type != ST_VARIABLE)
                error("Not a variable");
            return (sym->s_value);
        }
        senter(name,ST_VARIABLE,++vcnt);
        return (vcnt);
    }

    /* penter - enter a property into the symbol table */
    int penter(name)
      char *name;
    {
        SYMBOL *sym;

        if (sym = sfind(name)) {
            if (sym->s_type != ST_PROPERTY)
                error("Not a property");
            return (sym->s_value);
        }
        senter(name,ST_PROPERTY,++pcnt);
        return (pcnt);
    }

    /* center - enter a constant into the symbol table */
    center(name,value)
      char *name; int value;
    {
        if (sfind(name)) {
            error("Already defined");
            return;
        }
        senter(name,ST_CONSTANT,value);
    }

    /* sfind - find a symbol in the symbol table */
    SYMBOL *sfind(name)
      char *name;
    {
        SYMBOL *sym;
        for (sym = symbols; sym != NULL; sym = sym->s_next)
            if (strcmp(name,sym->s_name) == 0)
                break;
        return (sym);
    }

    /* senter - enter a symbol into the symbol table */
    SYMBOL *senter(name,type,value)
      char *name; int type,value;
    {
        SYMBOL *sym;

        if ((sym = (SYMBOL *)malloc(sizeof(SYMBOL))) == NULL)
            fail("out of memory");
        sym->s_name = save(name);
        sym->s_type = type;
        sym->s_value = value;
        sym->s_next = symbols;
        symbols = sym;
        return (sym);
    }
```

```
    /* frequire - fetch a token and check it */
    frequire(rtkn)
      int rtkn;
    {
        require(token(),rtkn);
    }

    /* require - check for a required token */
    require(tkn,rtkn)
      int tkn,rtkn;
    {
        char msg[100];
    if (tkn != rtkn) {
        sprintf(msg,"Expecting %s",t_names[rtkn]);
        error(msg);
    }
}

/* save - allocate memory for a string */
char *save(str)
  char *str;
{
    char *new;

    if ((new = malloc(strlen(str)+1)) == NULL)
        fail("out of memory");
    strcpy(new,str);
    return (new);
}

/* match - compare a string with the current token */
int match(str)
  char *str;{
    return (strcmp(str,t_token) == 0);
}

/* fail - print an error message and exit */
fail(msg)
  char *msg;
{
    printf("%s\n",msg);
#ifdef MAC
    macpause();
#endif
    exit();
}
```

ADVFCN.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advfcn.c - functions for the adventure compiler */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advcom.h"
#include "advdbs.h"

/* external variables */
extern char aname[];            /* adventure name */
extern int aversion;            /* adventure version number */
extern int cptr;                /* code space pointer */
extern int objbuf[];            /* object staging buffer */
extern int nprops;              /* number of properties in current object */
extern int t_value;             /* token value */
extern char t_token[];          /* token string */
extern char *t_names[];         /* token names */
```

continued

```
extern int otable[];                /* object table */
extern int curobj;                  /* current object number */
extern int curact;                  /* current action offset */
extern int atable[],acnt;           /* action table and count */
extern ARGUMENT *arguments;         /* function argument list */
extern ARGUMENT *temporaries;       /* function temporary variable list */
extern int def_flag;                /* default action flag value */
extern int def_mask;                /* default action mask value */

/* external routines */
extern char *malloc();
extern char *save();

/* do_adventure - handle the <ADVENTURE name version-number> statement */
do_adventure()
{    /* get the adventure name */
    frequire(T_IDENTIFIER);
    strncpy(aname,t_token,18);
    aname[18] = 0;

    /* get the adventure version number */
    frequire(T_NUMBER);
    aversion = t_value;

    /* check for the closing paren */
    frequire(T_CLOSE);
}

/* do_word - enter words of a particular type */
do_word(type)
{
    int tkn;

    while ((tkn = token()) == T_IDENTIFIER)
        add_word(t_token,type);
    require(tkn,T_CLOSE);
}

/* do_synonym - handle the <SYNONYMS ... > statement */
do_synonym()
{
    int tkn,wrd;

    frequire(T_IDENTIFIER);
    wrd = add_word(t_token,WT_UNKNOWN);
    while ((tkn = token()) == T_IDENTIFIER)
        add_synonym(t_token,wrd);
    require(tkn,T_CLOSE);
}

/* do_define - handle the <DEFINE ... > statement */
do_define()
{
    char name[TKNSIZE+1];
    int tkn;

    if ((tkn = token()) == T_OPEN)
        return (do_function());
    stoken(tkn);

    while ((tkn = token()) == T_IDENTIFIER) {
        strcpy(name,t_token);
        center(name,getvalue());
    }
    require(tkn,T_CLOSE);
}

/* do_variable - handle the <VARIABLE ... > statement */
do_variable()
{    int tkn;

    while ((tkn = token()) == T_IDENTIFIER)
        venter(t_token);
    require(tkn,T_CLOSE);
}
```

```
/* do_defproperty - handle the <PROPERTY ... > statement */
do_defproperty()
{
    int tkn;

    while ((tkn = token()) == T_IDENTIFIER)
        penter(t_token);
    require(tkn,T_CLOSE);
}

/* do_default - handle the <DEFAULT ... > statement */
do_default()
{
    int tkn;

    /* process statements until end of file */
    while ((tkn = token()) == T_OPEN) {
        frequire(T_IDENTIFIER);
        if (match("actor"))
            do_dflag(A_ACTOR);
        else if (match("direct-object"))
            do_dflag(A_DOBJECT);
        else if (match("indirect-object"))
            do_dflag(A_IOBJECT);
        else
            error("Unknown default definition statement type");
    }
    require(tkn,T_CLOSE);
    }

    /* do_dflag - handle ACTOR, DIRECT-OBJECT, and INDIRECT-OBJECT statements */
    do_dflag(flag)
      int flag;
    {
        int tkn;

        if ((tkn = token()) == T_IDENTIFIER) {
            if (match("required")) {
                def_flag |= flag;
                def_mask &= flag;
            }
            else if (match("forbidden")) {
                def_flag &= flag;
                def_mask &= flag;
            }
            else if (match("optional"))
                def_mask |= flag;
            else            error("Expecting: REQUIRED, FORBIDDEN or OPTIONAL");
            tkn = token();
        }
        else {
            def_flag |= flag;
            def_mask &= flag;
        }
        require(tkn,T_CLOSE);
    }

    /* do_object - handle object (LOCATION,OBJECT,ACTOR) definitions */
    int do_object(cname,class)
      char *cname; int class;
    {
        int tkn,obj,obase,osize,i,p;

    printf("[ %s: ",cname);
        frequire(T_IDENTIFIER);
    printf("%s ]\n",t_token);
        obj = curobj = oenter(t_token);

        /* initialize the object */
        objbuf[O_CLASS/2] = class;
        objbuf[O_NOUNS/2] = NIL;
        objbuf[O_ADJECTIVES/2] = NIL;
        objbuf[O_NPROPERTIES/2] = nprops = 0;
```

continued

```
/* copy the property list of the class object */
if (class) {
    obase = otable[class];
    osize = getword(obase+O_NPROPERTIES);
    for (i = p = 0; i < osize; i++, p += 4)
        if ((getword(obase+O_PROPERTIES+p) & P_CLASS) == 0)
            addprop(getword(obase+O_PROPERTIES+p),0,
                    getword(obase+O_PROPERTIES+p+2));
}

/* process statements until end of file */
while ((tkn = token()) == T_OPEN) {
    frequire(T_IDENTIFIER);
    if (match("noun"))
        do_noun();
    else if (match("adjective"))
        do_adjective();
    else if (match("property"))
        do_property(0);
    else if (match("class-property"))
        do_property(P_CLASS);
    else if (match("method"))
        do_method();
    else
        error("Unknown object definition statement type");
}
require(tkn,T_CLOSE);
/* copy the object to data memory */
osize = O_SIZE/2 + nprops*2;
obase = dalloc(osize*2);
for (i = p = 0; i < osize; i++, p += 2)
    putword(obase+p,objbuf[i]);
otable[obj] = obase;
curobj = NIL;

/* return the object number */
return (obj);
}

/* do_noun - handle the <NOUN ... > statement */
do_noun()
{
    int tkn,new;

    while ((tkn = token()) == T_IDENTIFIER) {
        new = dalloc(L_SIZE);
        putword(new+L_DATA,add_word(t_token,WT_NOUN));
        putword(new+L_NEXT,objbuf[O_NOUNS/2]);
        objbuf[O_NOUNS/2] = new;
    }
    require(tkn,T_CLOSE);
}

/* do_adjective - handle the <ADJECTIVE ... > statement */
do_adjective()
{
    int tkn,new;

    while ((tkn = token()) == T_IDENTIFIER) {
        new = dalloc(L_SIZE);
        putword(new+L_DATA,add_word(t_token,WT_ADJECTIVE));
        putword(new+L_NEXT,objbuf[O_ADJECTIVES/2]);
        objbuf[O_ADJECTIVES/2] = new;
    }
    require(tkn,T_CLOSE);
}

/* do_property - handle the <PROPERTY ... > statement */
do_property(flags)
  int flags;
{
    int tkn,name,value;

    while ((tkn = token()) == T_IDENTIFIER || tkn == T_NUMBER) {
        name = (tkn == T_IDENTIFIER ? penter(t_token) : t_value);
```

```
        value = getvalue();
        setprop(name,flags,value);
    }
    require(tkn,T_CLOSE);
}

/* do_method - handle <METHOD (FUN ...) ... > statement */do_method()
{
    int tkn,name,tcnt;

    /* get the property name */
    require(T_OPEN);
    require(T_IDENTIFIER);
printf("[ method: %s ]\n",t_token);

    /* create a new property */
    name = penter(t_token);

    /* allocate a new (anonymous) action */
    if (acnt < AMAX)
        ++acnt;
    else
        error("too many actions");

    /* store the action as the value of the property */
    setprop(name,P_CLASS,acnt);

    /* initialize the action */
    curact = atable[acnt] = dalloc(A_SIZE);
    putword(curact+A_VERBS,NIL);
    putword(curact+A_PREPOSITIONS,NIL);
    arguments = temporaries = NULL;
    tcnt = 0;

    /* enter the "self" argument */
    addargument(&arguments,"self");
    addargument(&arguments,"(dummy)");

    /* get the argument list */
    while ((tkn = token()) != T_CLOSE) {
        require(tkn,T_IDENTIFIER);
        if (match("&aux"))
            break;
        addargument(&arguments,t_token);
    }
    /* check for temporary variable definitions */
    if (tkn == T_IDENTIFIER)
        while ((tkn = token()) != T_CLOSE) {
            require(tkn,T_IDENTIFIER);
            addargument(&temporaries,t_token);
            tcnt++;
        }

    /* store the code address */
    putword(curact+A_CODE,cptr);

    /* allocate space for temporaries */
    if (temporaries) {
        putcbyte(OP_TSPACE);
        putcbyte(tcnt);    }

    /* compile the code */
    do_code(NULL);

    /* free the argument and temporary variable symbol tables */
    freelist(arguments);
    freelist(temporaries);
    arguments = temporaries = NULL;
}

/* setprop - set the value of a property */
setprop(prop,flags,value)
  int prop,flags,value;
{
    int i;
```

continued

```
    /* look for the property */
    for (i = 0; i < nprops; i++)
        if ((objbuf[O_PROPERTIES/2 + i*2] & P_CLASS) == prop) {
            objbuf[O_PROPERTIES/2 + i*2 + 1] = value;
            return;
        }
    addprop(prop,flags,value);
}

/* addprop - add a property to the current object's property list */
addprop(prop,flags,value)
  int prop,flags,value;
{
    if (nprops >= OPMAX) {
        printf("too many properties for this object\n");
        return;
    }
    objbuf[O_PROPERTIES/2 + nprops*2] = prop|flags;
    objbuf[O_PROPERTIES/2 + nprops*2 + 1] = value;
    objbuf[O_NPROPERTIES/2] = ++nprops;
}

/* do_code - compile code for an expression */
int do_code(type)
  char *type;
{
    int adr,tkn;

    if (type) printf("[ compiling %s code ]\n",type);
    adr = putcbyte(OP_PUSH);
    while ((tkn = token()) != T_CLOSE) {
        stoken(tkn);
        do_expr();
    }
    putcbyte(OP_RETURN);
    return (adr);
}
/* do_action - handle <ACTION ... > statement */
do_action()
{
    int tkn,act;

    /* get the action name */
    frequire(T_IDENTIFIER);
printf("[ action: %s ]\n",t_token);

    /* create a new action */
    act = aenter(t_token);
    curact = atable[act] = dalloc(A_SIZE);
    putword(curact+A_VERBS,NIL);
    putword(curact+A_PREPOSITIONS,NIL);
    putbyte(curact+A_FLAG,def_flag);
    putbyte(curact+A_MASK,def_mask);
    putword(curact+A_CODE,NIL);

    /* process statements until end of file */
    while ((tkn = token()) == T_OPEN) {
        frequire(T_IDENTIFIER);
        if (match("actor"))
            do_flag(A_ACTOR);
        else if (match("verb"))
            do_verb();
        else if (match("direct-object"))
            do_flag(A_DOBJECT);
        else if (match("preposition"))
            do_preposition();
        else if (match("indirect-object"))
            do_flag(A_IOBJECT);
        else if (match("code"))
            putword(curact+A_CODE,do_code(NULL));
        else
            error("Unknown action definition statement type");
    }
    require(tkn,T_CLOSE);
}
```

```
/* do_flag - handle ACTOR, DIRECT-OBJECT, and INDIRECT-OBJECT statements */
do_flag(flag)
  int flag;
{
    int tkn;

    if ((tkn = token()) == T_IDENTIFIER) {
        if (match("required")) {
            putbyte(curact+A_FLAG,getbyte(curact+A_FLAG) | flag);
            putbyte(curact+A_MASK,getbyte(curact+A_MASK) & flag);
        }
        else if (match("forbidden")) {
            putbyte(curact+A_FLAG,getbyte(curact+A_FLAG) & flag);
            putbyte(curact+A_MASK,getbyte(curact+A_MASK) & flag);
        }
        else if (match("optional"))
            putbyte(curact+A_MASK,getbyte(curact+A_MASK) | flag);
        else
            error("Expecting: REQUIRED, FORBIDDEN or OPTIONAL");
        tkn = token();
    }
    else {
        putbyte(curact+A_FLAG,getbyte(curact+A_FLAG) | flag);
        putbyte(curact+A_MASK,getbyte(curact+A_MASK) & flag);
    }
    require(tkn,T_CLOSE);
}

/* do_verb - handle the <VERB ... > statement */
do_verb()
{
    int tkn,new,lst;

    while ((tkn = token()) == T_IDENTIFIER || tkn == T_OPEN) {
        new = dalloc(L_SIZE);
        putword(new+L_NEXT,getword(curact+A_VERBS));
        putword(curact+A_VERBS,new);
        lst = dalloc(L_SIZE);
        putword(lst+L_NEXT,NIL);
        putword(new+L_DATA,lst);
        if (tkn == T_IDENTIFIER)
            putword(lst+L_DATA,add_word(t_token,WT_VERB));
        else {
            if ((tkn = token()) == T_IDENTIFIER)
                putword(lst+L_DATA,add_word(t_token,WT_VERB));
            else
                error("Expecting verb");
            while ((tkn = token()) == T_IDENTIFIER) {
                new = dalloc(L_SIZE);
                putword(new+L_DATA,add_word(t_token,WT_UNKNOWN));
                putword(new+L_NEXT,NIL);
                putword(lst+L_NEXT,new);
                lst = new;
            }
            require(tkn,T_CLOSE);
        }
    }
    require(tkn,T_CLOSE);
}

/* do_preposition - handle the <PREPOSITION ... > statement */
do_preposition()
{
    int tkn,new;

    while ((tkn = token()) == T_IDENTIFIER) {
        new = dalloc(L_SIZE);
        putword(new+L_DATA,add_word(t_token,WT_PREPOSITION));
        putword(new+L_NEXT,getword(curact+A_PREPOSITIONS));
        putword(curact+A_PREPOSITIONS,new);
    }    require(tkn,T_CLOSE);
}

/* do_function - handle <DEFINE (FUN ...) ... > statement */
do_function()
{
```

continued

```
    int tkn,act,tcnt;

    /* get the function name */
    frequire(T_IDENTIFIER);
printf("[ function: %s ]\n",t_token);

    /* create a new action */
    act = aenter(t_token);

    /* initialize the action */
    curact = atable[act] = dalloc(A_SIZE);
    putword(curact+A_VERBS,NIL);
    putword(curact+A_PREPOSITIONS,NIL);
    arguments = temporaries = NULL;
    tcnt = 0;

    /* get the argument list */
    while ((tkn = token()) != T_CLOSE) {
        require(tkn,T_IDENTIFIER);
        if (match("&aux"))
            break;
        addargument(&arguments,t_token);
    }

    /* check for temporary variable definitions */
    if (tkn == T_IDENTIFIER)
        while ((tkn = token()) != T_CLOSE) {
            require(tkn,T_IDENTIFIER);
            addargument(&temporaries,t_token);
            tcnt++;
        }

    /* store the code address */
    putword(curact+A_CODE,cptr);

    /* allocate space for temporaries */
    if (temporaries) {
        putcbyte(OP_TSPACE);
        putcbyte(tcnt);
    }

    /* compile the code */
    do_code(NULL);

    /* free the argument and temporary variable symbol tables */
    freelist(arguments);
    freelist(temporaries);
    arguments = temporaries = NULL;
}
/* addargument - add a formal argument */
addargument(list,name)
  ARGUMENT **list; char *name;
{
    ARGUMENT *arg;

    if ((arg = (ARGUMENT *)malloc(sizeof(ARGUMENT))) == NULL)
        fail("out of memory");
    arg->arg_name = save(name);
    arg->arg_next = *list;
    *list = arg;
}

/* freelist - free a list of arguments or temporaries */
freelist(arg)
  ARGUMENT *arg;
{
    ARGUMENT *nxt;
    while (arg) {
        nxt = arg->arg_next;
        free(arg->arg_name);
        free(arg);
        arg = nxt;
    }
}
```

```
/* findarg - find an argument offset */
int findarg(name)
  char *name;
{
    ARGUMENT *arg;
    int n;

    for (n = 0, arg = arguments; arg; n++, arg = arg->arg_next)
        if (strcmp(name,arg->arg_name) == 0)
            return (n);
    return (-1);
}

/* findtmp - find a temporary variable offset */
int findtmp(name)
  char *name;
{
    ARGUMENT *tmp;
    int n;

    for (n = 0, tmp = temporaries; tmp; n++, tmp = tmp->arg_next)
        if (strcmp(name,tmp->arg_name) == 0)
            return (n);
    return (-1);
}
```

ADVSCN.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advscn.c - a lexical scanner for the adventure compiler */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advcom.h"

/* useful definitions */
#define maplower(ch)    (isupper(ch) ? tolower(ch) : ch)

/* global variables */
int errcount=0;         /* error count */
int t_value;            /* numeric value */
char t_token[TKNSIZE+1];/* token string */
char *t_names[] = {
        0,
        "(",
        ")",
        "STRING",
        "IDENTIFIER",
        "NUMBER",
        "EOF"
};

/* external variables */
extern FILE *ifp;       /* input file pointer */
extern int msgoff;      /* message section offset */

/* local variables */
static int savetkn = 0; /* look ahead token */
static int savech = 0;  /* look ahead character */
static char fname[200]; /* include file name */
static char line[200];  /* current input line */
static char *lptr;      /* line pointer */
static int lnum;        /* line number */
static int ieof;        /* input end of file flag */
static int save_lnum;   /* saved lnum */
```

continued

```
static FILE *save_ifp;  /* saved ifp */
static int scnt;        /* count of characters in string */

/* sinit - initialize the scanner */
sinit()
{
    /* setup the line buffer */
    iptr = line; *iptr = 0;
    inum = 0;
    /* no include file yet */
    save_ifp = NULL;

    /* no lookahead yet */
    savech = 0;
    savetkn = 0;

    /* not eof yet */
    ieof = FALSE;
}

/* token - get the next token */
int token()
{
    int tkn;

    if (tkn = savetkn)
        savetkn = 0;
    else
        tkn = rtoken();
    return (tkn);
}

/* stoken - save a token */
stoken(tkn)
  int tkn;
{
    savetkn = tkn;
}

/* rtoken - read the next token */
int rtoken()
{
    int ch;

    /* check the next character */
    for (;;)
        switch (ch = skipspaces()) {
        case EOF:       return (T_EOF);
        case '(':       strcpy(t_token,"("); return (T_OPEN);
        case ')':       strcpy(t_token,")"); return (T_CLOSE);
        case '"':       return (getstring());
        case ';':       while (getch() != '\n'); break;
        default:        return (getid(ch));
        }
}

/* getstring - get a string */
int getstring()
{
    int ch,sflag;

    t_value = msgoff;
    sflag = FALSE;    scnt = 0;
    while ((ch = getch()) != EOF && ch != '"')
        if (isspace(ch))
            sflag = TRUE;
        else {
            if (ch == '\\')
                switch (ch = getch()) {
                case 'n':  ch = '\n'; break;
                case 't':  ch = '\t'; break;
                }
            if (sflag)
                { wputc(' '); sflag = FALSE; }
            wputc(ch);
        }
```

```
    if (sflag)
        wputc(' ');
    strdone();
    strcpy(t_token,"{string}");
    return (T_STRING);
}
/* getid - get an identifier */
int getid(ch)
  int ch;
{
    char *p;

    p = t_token; *p++ = maplower(ch);
    while ((ch = getch()) != EOF && isidchar(ch))
        *p++ = maplower(ch);
    *p = EOS;
    savech = ch;
    return (isnumber(t_token,&t_value) ? T_NUMBER : T_IDENTIFIER);
}

/* isnumber - check if this string is a number */
int isnumber(str,pval)
  char *str; int *pval;
{
    int digits;
    char *p;

    /* initialize */
    p = str; digits = 0;

    /* check for a sign */
    if (*p == '+' || *p == '-')
        p++;

    /* check for a string of digits */
    while (isdigit(*p))
        p++, digits++;

    /* make sure there was at least one digit and this is the end */
    if (digits == 0 || *p)        return (FALSE);

    /* convert the string to an integer and return successfully */
    if (*str == '+') ++str;
    *pval = atoi(str);
    return (TRUE);
}

/* wputc - put a character into the output file */
wputc(ch)
  int ch;
{
    ad_putc(encode(ch));
    scnt++;
}

/* strdone - finish a string */
strdone()
{
    wputc('\0');
    while (scnt & 3)
        wputc('\0');
    msgoff += scnt >> 2;
}

/* skipspaces - skip leading spaces */
skipspaces()
{
    int ch;

    while ((ch = getch()) && isspace(ch))
        ;
    return (ch);
}
```

continued

```
/* isidchar - is this an identifier character */
int isidchar(ch)
  int ch;
{
    return (!isspace(ch) && ch != '(' && ch != ')' && ch != '"');
}

/* getch - get the next character */
int getch()
{
    FILE *fp;
    int ch;

    /* check for a lookahead character */
    if (ch = savech)
       savech = 0;

   /* check for a buffered character */
   else if (ch = *lptr)
       lptr++;
   /* check for end of file */
   else if (ieof)
       ch = EOF;

   /* read another line */
   else {

       /* read the line */
       for (lptr = line; (ch = getchr()) != EOF && (*lptr++ = ch) != '\n'; )
           ;
       *lptr = 0;
       lnum++;

       /* check for an included file */
       if (line[0] == '@') {

           /* open the file */
           strcpy(fname,&line[1]); fname[strlen(fname)-1] = 0;
           if ((fp = fopen(fname,"r")) == NULL) {
               printf("Can't open include file: %s\n",fname);
               exit();
           }
           printf("[ including %s ]\n",fname);

           /* setup input from the file */
           save_lnum = lnum;
           save_ifp = ifp;
           ifp = fp;

           /* setup for the first line */
           lptr = line; *lptr = 0;
           lnum = 0;
       }

       /* otherwise this must be an input line */
       else {

           /* terminate the line with a newline */
           *lptr++ = '\n'; *lptr = 0;

           /* check for end of file */
           if (ch == EOF)
               ieof = TRUE;

           /* update the line number and setup for the new line */
           lptr = line;
       }

       /* get a character */
       ch = getch();
   }

   /* return the current character */
   return (ch);}
```

```
/* getchr - get a character checking for end of file */
int getchr()
{
    int ch;

    if ((ch = getc(ifp)) == EOF || ch == '\032') {
        if (save_ifp) {
            printf("[ end of %s ]\n",fname);
            fclose(ifp);
            lnum = save_lnum;
            ifp = save_ifp;
            save_ifp = NULL;
            ch = getchr();
        }
        else
            ch = EOF;
    }
    else if (ch == '\r')
        ch = getchr();
    return (ch);
}

/* encode - encode a single character */

int encode(ch)
  int ch;
{
    return ((ch - 30) & 0xFF);
}

/* error - report an error in the current line */
error(msg)
  char *msg;
{
    char *p;

    printf(">>> %s <<<\n>>> in line %d <<<\n%s",msg,lnum,line);
    for (p = line; p < lptr; p++)
        if (*p == '\t')
            putchar('\t');
        else
            putchar(' ');
    printf("^\n");
    errcount++;
#ifdef MAC
    macpause();
#endif
}

/* xerror - report an error in the current line */
xerror(msg)
  char *msg;
{
    printf(">>> %s <<<\n",msg);    errcount++;
}
```

ADVEXP.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advexp.c - expression compiler for adventure games */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advcom.h"
#include "advdbs.h"
```

continued

```
/* external routines */
extern SYMBOL *sfind();

/* external variables */
extern char t_token[];
extern int t_value;
extern int curobj;
extern char *code;
extern int cptr;

/* forward declarations */
int do_cond(),do_and(),do_or(),do_if(),do_while(),do_progn();
int do_setq(),do_return(),do_send(),do_sndsuper();

/* opcode tables */
static struct { char *nt_name; int nt_code,nt_args; } *nptr,ntab[] = {
        "not",          OP_NOT,         1,
        "+",            OP_ADD,         2,
        "-",            OP_SUB,         2,
        "*",            OP_MUL,         2,
        "/",            OP_DIV,         2,
        "%",            OP_REM,         2,
        "&",            OP_BAND,        2,
        "|",            OP_BOR,         2,
        "",             OP_BNOT,        1,
        "<",            OP_LT,          2,
        "=",            OP_EQ,          2,
        ">",            OP_GT,          2,
        "getp",         OP_GETP,        2,
        "setp",         OP_SETP,        3,
        "class",        OP_CLASS,       1,
        "match",        OP_MATCH,       2,
        "print",        OP_PRINT,       1,
        "print-number", OP_PNUMBER,     1,
        "print-noun",   OP_PNOUN,       1,
        "terpri",       OP_TERPRI,      0,
        "finish",       OP_FINISH,      0,        "chain",        OP_CHAIN,       0,
        "abort",        OP_ABORT,       0,
        "exit",         OP_EXIT,        0,
        "save",         OP_SAVE,        0,
        "restore",      OP_RESTORE,     0,
        "restart",      OP_RESTART,     0,
        "yes-or-no",    OP_YORN,        0,
        "rand",         OP_RAND,        1,
        "randomize",    OP_RNDMIZE,     0,
        0
};
static struct { char *ft_name; int (*ft_fcn)(); } *fptr,ftab[] = {
        "cond",         do_cond,
        "and",          do_and,
        "or",           do_or,
        "if",           do_if,
        "while",        do_while,
        "progn",        do_progn,
        "setq",         do_setq,
        "return",       do_return,
        "send",         do_send,
        "send-super",   do_sndsuper,
        0
};

/* do_expr - compile a subexpression */
do_expr()
{
    int tkn;

    switch (token()) {
    case T_OPEN:
        switch (tkn = token()) {
        case T_IDENTIFIER:
            if (in_ntab() || in_ftab())
                break;
        default:
            stoken(tkn);
            do_call();
```

```
        }
        break;
    case T_NUMBER:
        do_literal();
        break;
    case T_STRING:
        do_literal();
        break;
    case T_IDENTIFIER:
        do_identifier();
        break;
    default:
        error("Expecting expression");
    }
}
/* in_ntab - check for a function in ntab */
int in_ntab()
{
    for (nptr = ntab; nptr->nt_name; ++nptr)
        if (strcmp(t_token,nptr->nt_name) == 0) {
            do_nary(nptr->nt_code,nptr->nt_args);
            return (TRUE);
        }
    return (FALSE);
}

/* in_ftab - check for a function in ftab */
int in_ftab()
{
    for (fptr = ftab; fptr->ft_name; ++fptr)
        if (strcmp(t_token,fptr->ft_name) == 0) {
            (*fptr->ft_fcn)();
            return (TRUE);
        }
    return (FALSE);
}

/* do_cond - compile the (COND ... ) expression */
do_cond()
{
    int tkn,nxt,end;
    /* initialize the fixup chain */
    end = NIL;

    /* compile each COND clause */
    while ((tkn = token()) != T_CLOSE) {
        require(tkn,T_OPEN);
        do_expr();
        putcbyte(OP_BRF);
        nxt = putcword(NIL);
        while ((tkn = token()) != T_CLOSE) {
            stoken(tkn);
            do_expr();
        }
        putcbyte(OP_BR);
        end = putcword(end);
        fixup(nxt,cptr);
    }

    /* fixup references to the end of statement */
    if (end)
        fixup(end,cptr);
    else
        putcbyte(OP_NIL);
}

/* do_and - compile the (AND ... ) expression */
do_and()
{    int tkn,end;

    /* initialize the fixup chain */
    end = NIL;
```

continued

```
    /* compile each expression */
    while ((tkn = token()) != T_CLOSE) {
        stoken(tkn);
        do_expr();
        putcbyte(OP_BRF);
        end = putcword(end);
    }

    /* fixup references to the end of statement */
    if (end)
        fixup(end,cptr);
    else
        putcbyte(OP_NIL);
}

/* do_or - compile the (OR ... ) expression */
do_or()
{
    int tkn,end;

    /* initialize the fixup chain */
    end = NIL;

    /* compile each expression */
    while ((tkn = token()) != T_CLOSE) {
        stoken(tkn);
        do_expr();
        putcbyte(OP_BRT);
        end = putcword(end);
    }

    /* fixup references to the end of statement */
    if (end)
        fixup(end,cptr);
    else
        putcbyte(OP_T);
}

/* do_if - compile the (IF ... ) expression */
do_if()
{
    int tkn,nxt,end;

    /* compile the test expression */
    do_expr();

    /* skip around the 'then' clause if the expression is false */
    putcbyte(OP_BRF);
    nxt = putcword(NIL);
    /* compile the 'then' clause */
    do_expr();
    /* compile the 'else' clause */
    if ((tkn = token()) != T_CLOSE) {
        putcbyte(OP_BR);
        end = putcword(NIL);
        fixup(nxt,cptr);
        stoken(tkn);
        do_expr();
        frequire(T_CLOSE);
        nxt = end;
    }

    /* handle the end of the statement */
    fixup(nxt,cptr);
}

/* do_while - compile the (WHILE ... ) expression */
do_while()
{
    int tkn,nxt,end;

    /* compile the test expression */
    nxt = cptr;
    do_expr();
```

```
    /* skip around the 'then' clause if the expression is false */
    putcbyte(OP_BRF);
    end = putcword(NIL);

    /* compile the loop body */
    while ((tkn = token()) != T_CLOSE) {
        stoken(tkn);
        do_expr();
    }

    /* branch back to the start of the loop */
    putcbyte(OP_BR);
    putcword(nxt);

    /* handle the end of the statement */
    fixup(end,cptr);
}

/* do_progn - compile the (PROGN ... ) expression */
do_progn()
{
    int tkn,n;

    /* compile each expression */
    for (n = 0; (tkn = token()) != T_CLOSE; ++n) {
        stoken(tkn);
        do_expr();
    }
    /* check for an empty statement list */
    if (n == 0)
        putcbyte(OP_NIL);
}

/* do_setq - compile the (SETQ v x) expression */
do_setq()
{
    char name[TKNSIZE+1];
    int n;

    /* get the symbol name */
    frequire(T_IDENTIFIER);
    strcpy(name,t_token);

    /* compile the value expression */
    do_expr();

    /* check for this being a local symbol */
    if ((n = findarg(name)) >= 0)
        code_setargument(n);
    else if ((n = findtmp(name)) >= 0)
        code_settemporary(n);
    else {
        n = venter(name);
        code_setvariable(n);
    }
    frequire(T_CLOSE);
}

/* do_return - handle the (RETURN [expr]) expression */
do_return()
{
    int tkn;

    /* look for a result expression */
    if ((tkn = token()) != T_CLOSE) {
        stoken(tkn);
        do_expr();
        frequire(T_CLOSE);
    }

    /* otherwise, default the result to nil */
    else
        putcbyte(OP_NIL);
```

continued

```
    /* insert the return opcode */
    putcbyte(OP_RETURN);
}

/* do_send - handle the (SEND obj msg [expr]...) expression */
do_send()
{
    /* start searching for the method at the object itself */    putcbyte(OP_NIL);

    /* compile the object expression */
    putcbyte(OP_PUSH);
    do_expr();

    /* call the general message sender */
    sender();
}

/* do_sndsuper - handle the (SEND-SUPER msg [expr]...) expression */
do_sndsuper()
{
    /* start searching for the method at the current class object */
    code_literal(curobj);

    /* pass the message to "self" */
    putcbyte(OP_PUSH);
    code_argument(findarg("self"));

    /* call the general message sender */
    sender();
}

/* sender - compile an expression to send a message to an object */
sender()
{
    int tkn,n;

    /* compile the selector expression */
    putcbyte(OP_PUSH);
    do_expr();

    /* compile each argument expression */
    for (n = 2; (tkn = token()) != T_CLOSE; ++n) {
        stoken(tkn);
        putcbyte(OP_PUSH);
        do_expr();
    }
    putcbyte(OP_SEND);
    putcbyte(n);
}

/* do_call - compile a function call */
do_call()
{
    int tkn,n;

    /* compile the function itself */
    do_expr();

    /* compile each argument expression */
    for (n = 0; (tkn = token()) != T_CLOSE; ++n) {
        stoken(tkn);
        putcbyte(OP_PUSH);        do_expr();
    }
    putcbyte(OP_CALL);
    putcbyte(n);
}

/* do_nary - compile nary operator expressions */
do_nary(op,n)
  int op,n;
{
    while (n--) {
        do_expr();
        if (n) putcbyte(OP_PUSH);
    }
```

```
    putcbyte(op);
    frequire(T_CLOSE);
}

/* do_literal - compile a literal */
do_literal()
{
    code_literal(t_value);
}

/* do_identifier - compile an identifier */
do_identifier()
{
    SYMBOL *sym;
    int n;

    if (match("t"))
        putcbyte(OP_T);
    else if (match("nil"))
        putcbyte(OP_NIL);
    else if ((n = findarg(t_token)) >= 0)
        code_argument(n);
    else if ((n = findtmp(t_token)) >= 0)
        code_temporary(n);
    else if (sym = sfind(t_token)) {
        if (sym->s_type == ST_VARIABLE)
            code_variable(sym->s_value);
        else
            code_literal(sym->s_value);
    }
    else
        code_literal(oenter(t_token));
}

/* code_argument - compile an argument reference */
code_argument(n)
  int n;
{
    putcbyte(OP_ARG);
    putcbyte(n);
}
/* code_setargument - compile a set argument reference */
code_setargument(n)
  int n;
{
    putcbyte(OP_ASET);
    putcbyte(n);
}

/* code_temporary - compile an temporary reference */
code_temporary(n)
  int n;
{
    putcbyte(OP_TMP);
    putcbyte(n);
}

/* code_settemporary - compile a set temporary reference */
code_settemporary(n)
  int n;
{
    putcbyte(OP_TSET);
    putcbyte(n);
}

/* code_variable - compile a variable reference */
code_variable(n)
  int n;
{
    if (n < 32)
        putcbyte(OP_XVAR+n);
    else if (n < 256)
        { putcbyte(OP_SVAR); putcbyte(n); }
```

continued

```
    else
        { putcbyte(OP_VAR); putcword(n); }
}

/* code_setvariable - compile a set variable reference */
code_setvariable(n)
  int n;
{
    if (n < 32)
        putcbyte(OP_XSET+n);
    else if (n < 256)
        { putcbyte(OP_SSET); putcbyte(n); }
    else
        { putcbyte(OP_SET); putcword(n); }
}

/* code_literal - compile a literal reference */
code_literal(n)
  int n;
{
    if (n >= 0 && n < 64)
        putcbyte(OP_XPLIT+n);    else if (n < 0 && n > -64)
        putcbyte(OP_XNLIT-n);
    else if (n >= 64 && n < 256)
        { putcbyte(OP_SPLIT); putcbyte(n); }
    else if (n <= -64 && n > -256)
        { putcbyte(OP_SNLIT); putcbyte(-n); }
    else
        { putcbyte(OP_LIT); putcword(n); }
}

/* do_op - insert an opcode and look for closing paren */
do_op(op)
  int op;
{
    putcbyte(op);
    frequire(T_CLOSE);
}

/* putcbyte - put a code byte into data space */
int putcbyte(b)
  int b;
{
    if (cptr < CMAX)
        code[cptr++] = b;
    else
        error("insufficient code space");
    return (cptr-1);
}

/* putcword - put a code word into data space */
int putcword(w)
  int w;
{
    putcbyte(w);
    putcbyte(w >> 8);
    return (cptr-2);
}

/* fixup - fixup a reference chain */
fixup(chn,val)
  int chn,val;
{
    int hval,nxt;

    /* store the value into each location in the chain */
    for (hval = val >> 8; chn != NIL; chn = nxt) {
        if (chn < 0 || chn > CMAX-2)
            return;
        nxt = (code[chn] & 0xFF) | (code[chn+1] << 8);
        code[chn] = val;
        code[chn+1] = hval;
    }
}
```

ADVAVL.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advavl.c - avl tree manipulation routines */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advavl.h"
#include "advdbs.h"

#define TRUE    1
#define FALSE   0
#define NULL    0

/* external routines */
extern char *save();
extern char *malloc();

/* external variables */
extern char *data;
extern int curwrd;
extern int dptr;

/* local variables */
static TREE *curtree;
static char thiskey[WRDSIZE+1];

/* tnew - allocate a new avl tree */
TREE *tnew()
{
    TREE *tree;

    /* allocate the tree structure */
    if ((tree = (TREE *)malloc(sizeof(TREE))) == NULL)
        return (NULL);

    /* initialize the new tree */
    tree->tr_root = NULL;
    tree->tr_cnt = 0;

    /* return the new tree */
    return (tree);
}

/* tenter - add an entry to an avl tree */
int tenter(tree,key)
  TREE *tree; char *key;
{
    int h;    curtree = tree;
    strncpy(thiskey,key,WRDSIZE); thiskey[WRDSIZE] = 0;
    return (tenter1(&tree->tr_root,&h));
}

/* tenter1 - internal insertion routine */
int tenter1(pnode,ph)
  TNODE **pnode; int *ph;
{
    TNODE *p,*q,*r;
    int val,c;

    /* check for the subtree being empty */
    if ((p = *pnode) == NULL) {
        if (p = (TNODE *)malloc(sizeof(TNODE))) {
            curtree->tr_cnt++;
            KEY(p) = save(thiskey);
            WORD(p) = curwrd;
            LLINK(p) = RLINK(p) = NULL;
            B(p) = 0;
            *pnode = p;
```

continued

```
    *ph = TRUE;
    return (WORD(p));
}
else {
    *ph = FALSE;
        return (NIL);
    }
}

/* otherwise, check for a match at this node */
else if ((c = strcmp(thiskey,KEY(p))) == 0) {
    *ph = FALSE;
    return (WORD(p));
}

/* otherwise, check the left subtree */
else if (c < 0) {
    val = tenter1(&LLINK(p),ph);
    if (*ph)
        switch (B(p)) {
        case 1:
            B(p) = 0;
            *ph = FALSE;
            break;
        case 0:
            B(p) = -1;
            break;
        case -1:
            q = LLINK(p);
            if (B(q) == -1) {
                LLINK(p) = RLINK(q);
                RLINK(q) = p;
                B(p) = 0;
                p = q;                  }
            else {
                r = RLINK(q);
                RLINK(q) = LLINK(r);
                LLINK(r) = q;
                LLINK(p) = RLINK(r);
                RLINK(r) = p;
                B(p) = (B(r) == -1 ?  1 : 0);
                B(q) = (B(r) ==  1 ? -1 : 0);
                p = r;
            }
            B(p) = 0;
            *pnode = p;
            *ph = FALSE;
            break;
        }
}

/* otherwise, check the right subtree */
else {
    val = tenter1(&RLINK(p),ph);
    if (*ph)
        switch (B(p)) {
        case -1:
            B(p) = 0;
            *ph = FALSE;
            break;
        case 0:
            B(p) = 1;
            break;
        case 1:
            q = RLINK(p);
            if (B(q) == 1) {
                RLINK(p) = LLINK(q);
                LLINK(q) = p;
                B(p) = 0;
                p = q;
            }
            else {
                r = LLINK(q);
                LLINK(q) = RLINK(r);
                RLINK(r) = q;
                RLINK(p) = LLINK(r);
```

```
                LLINK(r) = p;
                B(p) = (B(r) ==  1 ? -1 : 0);
                B(q) = (B(r) == -1 ?  1 : 0);
                p = r;
            }
            B(p) = 0;
            *pnode = p;
            *ph = FALSE;
            break;
        }
}
/* return the node found or inserted */
    return (val);
}

/* tfind - find an entry in an avl tree */
int tfind(tree,key)
  TREE *tree; char *key;
{
    strncpy(thiskey,key,WRDSIZE); thiskey[WRDSIZE] = 0;
    return (tfind1(tree->tr_root));
}

/* tfind1 - internal lookup routine */
int tfind1(node)
  TNODE *node;
{
    int c;

    /* check for the subtree being empty */
    if (node == NULL)
        return (NIL);

    /* otherwise, check for a match at this node */
    else if ((c = strcmp(thiskey,KEY(node))) == 0)
        return (WORD(node));

    /* otherwise, check the left subtree */
    else if (c < 0)
        return (tfind1(LLINK(node)));

    /* otherwise, check the right subtree */
    else
        return (tfind1(RLINK(node)));
}
```

ADVFIO.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advfio.c - file i/o routines for the adventure compiler */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#define BSIZE   8192

/* global variables */
long ad_foff;

/* external routines */
extern long lseek();
/* local variables */
static char buf[BSIZE];
static int boff;
static int fd;
```

continued

```
ad_create(name)
  char *name;
{
    /* create the file */
    if ((fd = creat(name,0666)) < 0)
        fail("can't create output file");

    /* initialize the buffer and file offset */
    ad_foff = 0L;
    boff = 0;
}

ad_close()
{
    ad_flush();
    close(fd);
}

ad_putc(ch)
  int ch;
{
    buf[boff++] = ch; ad_foff++;
    if (boff >= BSIZE)
        ad_flush();
}

ad_seek(pos)
  long pos;
{
    ad_flush();
    if (lseek(fd,pos,0) != pos)
        fail("error positioning output file");
    ad_foff = pos;
}

ad_flush()
{
    if (boff) {
        if (write(fd,buf,boff) != boff)
            fail("error writing to output file");
        boff = 0;
    }
}
```

ADVINT.H Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advint.h - adventure interpreter definitions */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include <stdio.h>
#include <ctype.h>

/* useful definitions */
#define TRUE            1
#define FALSE           0
#define EOS             '\0'

/* program limits */
#define STKSIZE         500

/* code completion codes */
#define FINISH          1
#define CHAIN           2
#define ABORT           3
```

ADVMSG.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advmsg.c - adventure interpreter message routines */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include <stdio.h>

/* cache size */
#define CSIZE   8

/* external routines */
extern char *malloc();

/* message block cache */
static char *mbuffer[CSIZE];    /* message text block cache buffers */
static int mblock[CSIZE];       /* message text block cache block numbers */
static int mnext[CSIZE];        /* next most recently used block */
static int mhead,mtail;         /* head and tail of lru list */

/* message file variables */
static int mbase;               /* message base block */
static int mfd;                 /* message file descriptor */

/* current message variables */
static int mblk;                /* current block */static char *mbuf;        /* current buffer */
static int moff;                /* current buffer offset */

/* msg_init - initialize the message routines */
msg_init(fd,base)
  int fd,base;
{
    char *p;
    int i;

    /* remember the message file descriptor and base */
    mbase = base;
    mfd = fd;

    /* initialize the cache */
    if ((p = malloc(CSIZE * 512)) == NULL)
        error("insufficient memory");
    for (i = 0; i < CSIZE; i++) {
        mbuffer[i] = p; p += 512;
        mblock[i] = -1;
        mnext[i] = i+1;
    }
    mhead = 0; mtail = CSIZE-1; mnext[mtail] = -1;
}

/* msg_open - open a message */
int msg_open(msg)
  unsigned int msg;
{
    /* save the current message block */
    mblk = msg >> 7;

    /* make sure the first block is in a buffer */
    get_block(mblk);

    /* setup the initial offset into the block */
    moff = (msg & 0x7F) << 2;
}

/* msg_byte - get a byte from a message */
int msg_byte()
{
```

continued

```
    /* check for end of block and get next block */
    if (moff >= 512) {
        get_block(++mblk);
        moff = 0;
    }

    /* return the next message byte */
    return (decode(mbuf[moff++]));
}

/* decode - decode a character */
int decode(ch)
  int ch;{
    return ((ch + 30) & 0xFF);
}

/* get_block - get a block of message text */
get_block(blk)
  unsigned int blk;
{
    int last,n;
    long loff;

    /* first check the cache */
    for (n = mhead; n != -1; last = n, n = mnext[n])
        if (blk == mblock[n]) {
            if (n != mhead) {
                if ((mnext[last] = mnext[n]) == -1)
                    mtail = last;
                mnext[n] = mhead;
                mhead = n;
            }
            mbuf = mbuffer[n];
            return;
        }

    /* overwrite the least recently used buffer */
    mblock[mtail] = blk;
    loff = ((long) mbase + (long) blk) << 9;
    lseek(mfd,loff,0);
    if (read(mfd,mbuffer[mtail],512) != 512)
        error("error reading message text");

    /* get the block */
    get_block(blk);
}
```

ADVTRM.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advtrm.c - terminal i/o routines */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include <stdio.h>

/* useful definitions */
#define TRUE    1
#define FALSE   0
#define EOS     '\0'
#define LINEMAX 200#define WORDMAX 100

/* global variables */
char line[LINEMAX+1];

/* local variables */
static int col,maxcol,row,maxrow;
```

```
static int scnt,wcnt;
static char word[WORDMAX+1],*wptr;
static FILE *logfp = NULL;

/* forward declarations */
char *trm_line();

/* trm_init - initialize the terminal module */
trm_init(rows,cols,name)
  int rows,cols; char *name;
{
    /* initialize the terminal i/o variables */
    maxcol = cols-1; col = 0;
    maxrow = rows-1; row = 0;
    wptr = word; wcnt = 0;
    scnt = 0;

    /* open the log file */
    if (name && (logfp = fopen(name,"w")) == NULL)
        error("can't open log file");
}

/* trm_done - finish terminal i/o */
trm_done()
{
    if (wcnt) trm_word();
    if (logfp) fclose(logfp);
}

/* trm_get - get a line */
char *trm_get(line)
  char *line;
{
    if (wcnt) trm_word();
    while (scnt--) putchr(' ');
    row = col = scnt = 0;
    return (trm_line(line));
}

/* trm_str - output a string */
trm_str(str)
  char *str;
{
    while (*str)
        trm_chr(*str++);
}

/* trm_xstr - output a string without logging or word wrap */
trm_xstr(str)
  char *str;
{
    while (*str)
        putch(*str++,stdout);
}

/* trm_chr - output a character */
trm_chr(ch)
  int ch;
{
    switch (ch) {
    case ' ':
            if (wcnt)
                trm_word();
            scnt++;
            break;
    case '\t':
            if (wcnt)
                trm_word();
            scnt = (col + 8) & 7;
            break;
    case '\n':
            if (wcnt)
                trm_word();
            trm_eol();
```

continued

```
            scnt = 0;
            break;
    default:
            if (wcnt < WORDMAX) {
                *wptr++ = ch;
                wcnt++;
            }
            break;
    }
}

/* trm_word - output the current word */
trm_word()
{
    if (col + scnt + wcnt > maxcol)
        trm_eol();
    else
        while (scnt--)
            { putchr(' '); col++; }
    for (wptr = word; wcnt--; col++)
        putchr(*wptr++);
    wptr = word;
    wcnt = 0;
    scnt = 0;
}

/* trm_eol - end the current line */
trm_eol()
{    putchr('\n');
    if (++row >= maxrow)
        { trm_wait(); row = 0; }
    col = 0;
}

/* trm_wait - wait for the user to type return */
trm_wait()
{
    trm_xstr("  << MORE >>\r");
    waitch();
    trm_xstr("            \r");
}

/* trm_line - get an input line */
char *trm_line(line)
  char *line;
{
    char *p;
    int ch;

    p = line;
    while ((ch = getchr()) != EOF && ch != '\n')
        switch (ch) {
        case '\177':
        case '\010':
                if (p != line) {
                    if (ch != '\010') putchr('\010',stdout);
                    putchr(' ',stdout);
                    putchr('\010',stdout);
                    p--;
                }
                break;
        default:
                if ((p - line) < LINEMAX)
                    *p++ = ch;
                break;
        }
    *p = 0;
    return (ch == EOF ? NULL : line);
}

/* getchr - input a single character */
int getchr()
{
    int ch;
```

```
    if ((ch = getch()) != EOF && logfp)
        putch(ch,logfp);
    return (ch);
}

/* putchr - output a single character */
putchr(ch)
  int ch;{
    if (logfp) putch(ch,logfp);
    putch(ch,stdout);
}
```

ADVPRS.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advprs.c - adventure parser */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advint.h"
#include "advdbs.h"

/* parser result variables */
int nouns[20];
int *adjectives[20];
static int actor,action,dobject,ndobjects,iobject;
static int flag;

/* external routines */
extern char *trm_get();

/* external variables */
extern char line[];     /* line buffer */

/* local variables */
static char *lptr;      /* line pointer */
static int words[100];  /* word table */
static char *wtext[100];/* word text table */
static int *wptr;       /* word pointer */
static int wcnt;        /* word count */

static int verbs[3];    /* words in the verb phrase */
static int nnums[20];   /* noun word numbers */
static int nptr;        /* noun pointer (actually, an index) */
static int adjs[100];   /* adjective lists */
static int anums[100];  /* adjective word numbers */
static int aptr;        /* adjective pointer (actually, an index) */

/* parse - read and parse an input line */
int parse()
{
    if (!parse1())
        return (FALSE);
    setvalue(V_ACTOR,actor);
    setvalue(V_ACTION,action);
    setvalue(V_DOBJECT,dobject);
    setvalue(V_NDOBJECTS,ndobjects);
    setvalue(V_IOBJECT,iobject);
    return (TRUE);
}

/* next - get the next command (next direct object) */
int next()
{
    if (getvalue(V_NDOBJECTS) > 1) {
        setvalue(V_ACTOR,actor);
        setvalue(V_ACTION,action);
```

continued

```
        setvalue(V_DOBJECT,getvalue(V_DOBJECT) + 1);
        setvalue(V_NDOBJECTS,getvalue(V_NDOBJECTS) - 1);
        setvalue(V_IOBJECT,iobject);
        return (TRUE);
    }
    else
        return (FALSE);
}

/* parse1 - the main parser */
int parse1()
{
    int noun1,cnt1,noun2,cnt2;
    int preposition,flag;

    /* initialize */
    noun1 = noun2 = NIL; cnt1 = cnt2 = 0;
    nptr = aptr = 0;
    preposition = 0;
    flag = 0;

    /* initialize the parser result variables */
    actor = action = dobject = iobject = NIL;
    ndobjects = 0;

    /* get an input line */
    if (!get_line())
        return (FALSE);

    /* check for actor */
    if (wtype(*wptr) == WT_ADJECTIVE || wtype(*wptr) == WT_NOUN) {
        if ((actor = getnoun()) == NIL)
            return (FALSE);
        flag |= A_ACTOR;
    }

    /* get verb phrase */
    if (!getverb())
        return (FALSE);

    /* direct object, preposition and indirect object */
    if (*wptr) {

        /* get the first set of noun phrases (direct objects) */
        noun1 = nptr+1;         for (;;) {

            /* get the next direct object */
            if (getnoun() == NIL)
                return (FALSE);
            ++cnt1;

            /* check for more direct objects */
            if (*wptr == NIL || wtype(*wptr) != WT_CONJUNCTION)
                break;
            wptr++;
        }

        /* get the preposition and indirect object */
        if (*wptr) {

            /* get the preposition */
            if (wtype(*wptr) == WT_PREPOSITION)
                preposition = *wptr++;

            /* get the second set of noun phrases (indirect object) */
            noun2 = nptr+1;
            for (;;) {

                /* get the next direct object */
                if (getnoun() == NIL)
                    return (FALSE);
                ++cnt2;

                /* check for more direct objects */
                if (*wptr == NIL || wtype(*wptr) != WT_CONJUNCTION)
                    break;
```

```
                wptr++;
            }
        }

        /* make sure this is the end of the sentence */
        if (*wptr) {
            parse_error();
            return (FALSE);
        }
    }

    /* setup the direct and indirect objects */
    if (preposition) {
        if (cnt2 > 1) {
            parse_error();
            return (FALSE);
        }
        dobject = noun1;
        ndobjects = cnt1;
        iobject = noun2;
    }
    else if (noun2) {
        if (cnt1 > 1) {
            parse_error();
            return (FALSE);
        }
        preposition = findword("to");
        dobject = noun2;
        ndobjects = cnt2;
        iobject = noun1;
    }
    else {
        dobject = noun1;
        ndobjects = cnt1;
    }

    /* setup the flags for the action lookup */
    if (dobject) flag |= A_DOBJECT;
    if (iobject) flag |= A_IOBJECT;

    /* find the action */
    if ((action = findaction(verbs,preposition,flag)) == NIL) {
        parse_error();
        return (FALSE);
    }

    /* return successfully */
    return (TRUE);
}

/* getverb - get a verb phrase and return the action it refers to */
int getverb()
{
    /* get the verb */
    if (*wptr == NIL || wtype(*wptr) != WT_VERB) {
        parse_error();
        return (NIL);
    }
    verbs[0] = *wptr++;
    verbs[1] = NIL;

    /* check for a word following the verb */
    if (*wptr) {
        verbs[1] = *wptr;
        verbs[2] = NIL;
        if (checkverb(verbs))
            wptr++;
        else {
            verbs[1] = words[wcnt-1];
            if (checkverb(verbs))
                words[--wcnt] = NIL;
            else {
                verbs[1] = NIL;
                if (!checkverb(verbs)) {
```

continued

```
                    parse_error();
                    return (NIL);
                }
            }       }
    }
    return (T);
}

/* getnoun - get a noun phrase and return the object it refers to */
int getnoun()
{
    /* initialize the adjective list pointer */
    adjectives[nptr] = adjs + aptr;

    /* get the optional article */
    if (*wptr != NIL && wtype(*wptr) == WT_ARTICLE)
        wptr++;

    /* get optional adjectives */
    while (*wptr != NIL && wtype(*wptr) == WT_ADJECTIVE) {
        adjs[aptr] = *wptr++;
        anums[aptr] = wptr - words - 1;
        aptr++;
    }
    adjs[aptr++] = NULL;

    /* get the noun itself */
    if (*wptr == NIL || wtype(*wptr) != WT_NOUN) {
        parse_error();
        return (NIL);
    }

    /* save the noun */
    nouns[nptr] = *wptr++;
    nnums[nptr] = wptr - words - 1;
    return (++nptr);
}

/* get_line - get the input line and lookup each word */
int get_line()
{
    /* read an input line */
    trm_chr(':');
    if ((iptr = trm_get(line)) == NULL) {
        trm_str("Speak up!  I can't hear you!\n");
        return (FALSE);
    }

    /* get each word on the line */
    for (wcnt = 0; skip_spaces(); wcnt++)
        if (get_word() == NIL)
            return (FALSE);
    words[wcnt] = NIL;

    /* check for a blank line */
    if (wcnt == 0) {
        trm_str("Speak up!  I can't hear you!\n");
        return (FALSE);    }

    /* point to the first word and return successfully */
    wptr = words;
    return (TRUE);
}

/* skip_spaces - skip leading spaces */
int skip_spaces()
{
    while (spacep(*iptr))
        iptr++;
    return (*iptr != EOS);
}

/* show_noun - show a noun phrase */
show_noun(n)
  int n;
{
```

```
int adj,*p;

/* print the adjectives */
for (p = adjectives[n-1], adj = FALSE; *p; p++, adj = TRUE) {
    if (adj) trm_chr(' ');
    trm_str(wtext[anums[p-adjs]]);
}

/* print the noun */
    if (adj) trm_chr(' ');
    trm_str(wtext[nnums[n-1]]);
}

/* get_word - get the next word */
int get_word()
{
    int ch;

    /* get the next word */
    for (wtext[wcnt] = lptr; (ch = *lptr) != EOS && !spacep(ch); )
        *lptr++ = (isupper(ch) ? tolower(ch) : ch);
    if (*lptr != EOS) *lptr++ = EOS;

    /* look up the word */
    if (words[wcnt] = findword(wtext[wcnt]))
        return (words[wcnt]);
    else {
        trm_str("I don't know the word \"");
        trm_str(wtext[wcnt]);
        trm_str("\".\n");
        return (NIL);
    }
}

/* spacep - is this character a space? */
int spacep(ch)  int ch;
{
    return (ch == ' ' || ch == ',' || ch == '.');
}

/* parse_error - announce a parsing error */
parse_error()
{
    trm_str("I don't understand.\n");
}
```

ADVDBS.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advdbs.c - adventure database access routines */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advint.h"
#include "advdbs.h"
#ifdef MAC
#include <fnctl.h>
#define RMODE   (O_RDONLY|O_BINARY)
#else
#include <setjmp.h>
#define RMODE   0
#endif

/* global variables */
int h_init;     /* initialization code */
```

continued

```
int h_update;    /* update code */
int h_before;    /* before handler code */
int h_after;     /* after handler code */
int h_error;     /* error handling code */
int datafd;      /* data file descriptor */

/* external variables */
extern jmp_buf restart;

/* external routines */
extern char *malloc();

/* table base addresses */
char *wtable;    /* word table */
char *wtypes;    /* word type table */
int wcount;      /* number of words */
char *otable;    /* object table */
int ocount;      /* number of objects */
char *atable;    /* action table */
int acount;      /* number of actions */
char *vtable;    /* variable table */
int vcount;      /* number of variables */
char *data;      /* base of data tables */
char *base;      /* current base address */
char *dbase;     /* base of the data space */
char *cbase;     /* base of the code space */
int length;      /* length of resident data structures */

/* data file header */
static char hdr[HDR_SIZE];

/* save parameters */
static long saveoff;     /* save data file offset */
static char *save;       /* save area base address */
static int slen;         /* save area length */

/* db_init - read and decode the data file header */
db_init(name)
  char *name;
{
    int woff,ooff,aoff,voff,n;
    char fname[50];

    /* get the data file name */
    strcpy(fname,name);
#ifndef MAC
    strcat(fname,".dat");
#endif

    /* open the data file */
    if ((datafd = open(fname,RMODE)) == -1)
        error("can't open data file");

    /* read the header */
    if (read(datafd,hdr,HDR_SIZE) != HDR_SIZE)
        error("bad data file");
    complement(hdr,HDR_SIZE);
    base = hdr;

    /* check the magic information */
    if (strncmp(&hdr[HDR_MAGIC],"ADVSYS",6) != 0)
        error("not an adventure data file");

    /* check the version number */
    if ((n = getword(HDR_VERSION)) < 101 || n > VERSION)
        error("wrong version number");

    /* decode the resident data length header field */
    length = getword(HDR_LENGTH);

    /* allocate space for the resident data structure */
    if ((data = malloc(length)) == 0)
        error("insufficient memory");
    /* compute the offset to the data */
    saveoff = (long)getword(HDR_DATBLK) * 512L;
```

```
    /* read the resident data structure */
    lseek(datafd,saveoff,0);
    if (read(datafd,data,length) != length)
        error("bad data file");
    complement(data,length);

    /* get the table base addresses */
    wtable = data + (woff = getword(HDR_WTABLE));
    wtypes = data + getword(HDR_WTYPES) - 1;
    otable = data + (ooff = getword(HDR_OTABLE));
    atable = data + (aoff = getword(HDR_ATABLE));
    vtable = data + (voff = getword(HDR_VTABLE));

    /* get the save data area */
    saveoff += (long)getword(HDR_SAVE);
    save = data + getword(HDR_SAVE);
    slen = getword(HDR_SLEN);

    /* get the base of the data and code spaces */
    dbase = data + getword(HDR_DBASE);
    cbase = data + getword(HDR_CBASE);

    /* initialize the message routines */
    msg_init(datafd,getword(HDR_MSGBLK));
    /* get the code pointers */
    h_init = getword(HDR_INIT);
    h_update = getword(HDR_UPDATE);
    h_before = getword(HDR_BEFORE);
    h_after = getword(HDR_AFTER);
    h_error = getword(HDR_ERROR);

    /* get the table lengths */
    base = data;
    wcount = getword(woff);
    ocount = getword(ooff);
    acount = getword(aoff);
    vcount = getword(voff);

    /* setup the base of the resident data */
    base = dbase;

    /* set the object count */
    setvalue(V_OCOUNT,ocount);
}

/* db_save - save the current database */
db_save(name)
  char *name;
{
    return (advsave(&hdr[HDR_ANAME],20,save,slen) ? T : NIL);}

/* db_restore - restore a saved database */
int db_restore(name)
  char *name;
{
    return (advrestore(&hdr[HDR_ANAME],20,save,slen) ? T : NIL);
}

/* db_restart - restart the current game */
db_restart()
{
    lseek(datafd,saveoff,0);
    if (read(datafd,save,slen) != slen)
        return (NIL);
    complement(save,slen);
    setvalue(V_OCOUNT,ocount);
    longjmp(restart,1);
}

/* complement - complement a block of memory */
complement(adr,len)
  char *adr; int len;
{
```

continued

```
    for (; len--; adr++)
        *adr = (*adr + 30);
}

/* findword - find a word in the dictionary */
int findword(word)
  char *word;
{
    char sword[WRDSIZE+1];
    int wrd,i;

    /* shorten the word */
    strncpy(sword,word,WRDSIZE); sword[WRDSIZE] = 0;

    /* look up the word */
    for (i = 1; i <= wcount; i++) {
        wrd = getwloc(i);
        if (strcmp(base+wrd+2,sword) == 0)
            return (getword(wrd));
    }
    return (NIL);
}

/* wtype - return the type of a word */
int wtype(wrd)
  int wrd;
{
    return (wtypes[wrd]);
}

/* match - match an object against a name and list of adjectives */
int match(obj,noun,adjs)
  int obj,noun,*adjs;
{
    int *aptr;

    if (!hasnoun(obj,noun))
        return (FALSE);
    for (aptr = adjs; *aptr != NIL; aptr++)
        if (!hasadjective(obj,*aptr))
            return (FALSE);
    return (TRUE);
}

/* checkverb - check to see if this is a valid verb */
int checkverb(verbs)
  int *verbs;
{
    int act;

    /* look up the action */
    for (act = 1; act <= acount; act++)
        if (hasverb(act,verbs))
            return (act);
    return (NIL);
}

/* findaction - find an action matching a description */
findaction(verbs,preposition,flag)
  int *verbs,preposition,flag;
{
    int act,mask;

    /* look up the action */
    for (act = 1; act <= acount; act++) {
        if (preposition && !haspreposition(act,preposition))
            continue;
        if (!hasverb(act,verbs))
            continue;
        mask = getabyte(act,A_MASK);
        if ((flag & mask) == (getabyte(act,A_FLAG) & mask))
            return (act);
    }
    return (NIL);
}
```

```
/* getp - get the value of an object property */
int getp(obj,prop)
  int obj,prop;
{
    int p;

    for (; obj; obj = getofield(obj,O_CLASS))
        if (p = findprop(obj,prop))
            return (getofield(obj,p));
    return (NIL);}

/* setp - set the value of an object property */
int setp(obj,prop,val)
  int obj,prop,val;
{
    int p;

    for (; obj; obj = getofield(obj,O_CLASS))
        if (p = findprop(obj,prop))
            return (putofield(obj,p,val));
    return (NIL);
}

/* findprop - find a property */
int findprop(obj,prop)
  int obj,prop;
{
    int n,i,p;

    n = getofield(obj,O_NPROPERTIES);
    for (i = p = 0; i < n; i++, p += 4)
        if ((getofield(obj,O_PROPERTIES+p) & P_CLASS) == prop)
            return (O_PROPERTIES+p+2);
    return (NIL);
}

/* hasnoun - check to see if an object has a specified noun */
int hasnoun(obj,noun)
  int obj,noun;
{
    while (obj) {
        if (inlist(getofield(obj,O_NOUNS),noun))
            return (TRUE);
        obj = getofield(obj,O_CLASS);
    }
    return (FALSE);
}

/* hasadjective - check to see if an object has a specified adjective */
int hasadjective(obj,adjective)
  int obj,adjective;
{
    while (obj) {
        if (inlist(getofield(obj,O_ADJECTIVES),adjective))
            return (TRUE);
        obj = getofield(obj,O_CLASS);
    }
    return (FALSE);
}

/* hasverb - check to see if this action has this verb */
int hasverb(act,verbs)
  int act,*verbs;
{    int link,word,*verb;

    /* get the list of verbs */
    link = getafield(act,A_VERBS);

    /* look for this verb */
    while (link != NIL) {
        verb = verbs;
        word = getword(link+L_DATA);
        while (*verb != NIL && word != NIL) {
            if (*verb != getword(word+L_DATA))
                break;
```

continued

```
            verb++;
            word = getword(word+L_NEXT);
        }
        if (*verb == NIL && word == NIL)
            return (TRUE);
        link = getword(link+L_NEXT);
    }
    return (FALSE);
}

/* haspreposition - check to see if an action has a specified preposition */
int haspreposition(act,preposition)
  int act,preposition;
{
    return (inlist(getafield(act,A_PREPOSITIONS),preposition));
}

/* inlist - check to see if a word is an element of a list */
int inlist(link,word)
  int link,word;
{
    while (link != NIL) {
        if (word == getword(link+L_DATA))
            return (TRUE);
        link = getword(link+L_NEXT);
    }
    return (FALSE);
}

/* getofield - get a field from an object */
int getofield(obj,off)
  int obj,off;
{
    return (getword(getoloc(obj)+off));
}

/* putofield - put a field into an object */
int putofield(obj,off,val)
  int obj,off,val;
{
    return (putword(getoloc(obj)+off,val));
}
/* getafield - get a field from an action */
int getafield(act,off)
  int act,off;
{
    return (getword(getaloc(act)+off));
}
/* getabyte - get a byte field from an action */
int getabyte(act,off)
  int act,off;
{
    return (getbyte(getaloc(act)+off));
}

/* getoloc - get an object from the object table */
int getoloc(n)
  int n;
{
    if (n < 1 || n > ocount)
        nerror("object number out of range: %d",n);
    return (getdword(otable+n+n));
}

/* getaloc - get an action from the action table */
int getaloc(n)
  int n;
{
    if (n < 1 || n > acount)
        nerror("action number out of range: %d",n);
    return (getdword(atable+n+n));
}
```

```
/* getvalue - get the value of a variable from the variable table */
int getvalue(n)
  int n;
{
    if (n < 1 || n > vcount)
        nerror("variable number out of range: %d",n);
    return (getdword(vtable+n+n));
}

/* setvalue - set the value of a variable in the variable table */
int setvalue(n,v)
  int n,v;
{
    if (n < 1 || n > vcount)
        nerror("variable number out of range: %d",n);
    return (putdword(vtable+n+n,v));
}

/* getwloc - get a word from the word table */
int getwloc(n)
  int n;
{
    if (n < 1 || n > wcount)
        nerror("word number out of range: %d",n);
    return (getdword(wtable+n+n));
}

/* getword - get a word from the data array */
int getword(n)
  int n;
{
    return (getdword(base+n));
}

/* putword - put a word into the data array */
int putword(n,w)
  int n,w;
{
    return (putdword(base+n,w));
}

/* getbyte - get a byte from the data array */
int getbyte(n)
  int n;
{
    return (*(base+n) & 0xFF);
}

/* getcbyte - get a code byte */
int getcbyte(n)
  int n;
{
    return (*(cbase+n) & 0xFF);
}

/* getcword - get a code word */
int getcword(n)
  int n;
{
    return (getdword(cbase+n));
}

/* getdword - get a word from the data array */
int getdword(p)
  char *p;
{
    return ((*p & 0xFF) | (*(p+1) << 8));
}

/* putdword - put a word into the data array */
int putdword(p,w)
  char *p; int w;
{
```

continued

```
    *p = w; *(p+1) = w >> 8;
    return (w);
}

/* nerror - handle errors with numeric arguments */
nerror(fmt,n)
  char *fmt; int n;
{
    char buf[100];
    sprintf(buf,fmt,n);
    error(buf);
}
```

ADVINT.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advint.c - an interpreter for adventure games */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advint.h"
#include "advdbs.h"
#ifndef MAC
#include <setjmp.h>
#endif

/* global variables */
jmp_buf restart;

/* external variables */
extern int h_init;
extern int h_update;
extern int h_before;
extern int h_after;
extern int h_error;

/* main - the main routine */
main(argc,argv)
  int argc; char *argv[];
{
    char *fname,*iname;
    int rows,cols,i;

#ifdef MAC
    char name[50];
    macinit(name);
    fname = name;
    iname = NULL;
    rows = 20;
    cols = 80;
#else
    printf("ADVINT v1.2 - Copyright (c) 1986, by David Betz\n");
    fname = NULL;
    iname = NULL;    rows = 24;
    cols = 80;

    /* parse the command line */
    for (i = 1; i < argc; i++)
        if (argv[i][0] == '-')
            switch (argv[i][1]) {
            case 'r':
            case 'R':
                    rows = atoi(&argv[i][2]);
                    break;
            case 'c':
            case 'C':
```

```
                    cols = atoi(&argv[i][2]);
                    break;
            case 'l':
            case 'L':
                    lname = &argv[i][2];
                    break;
            }
        else
            fname = argv[i];
    if (fname == NULL) {
        printf("usage: advint [-r<rows>] [-c<columns>] [-l<log-file>]
<file>\n");
        exit();
    }
#endif

    /* initialize terminal i/o */
    trm_init(rows,cols,lname);

    /* initialize the database */
    db_init(fname);

    /* play the game */
    play();
}

/* play - the main loop */
play()
{
    /* establish the restart point */
    setjmp(restart);

    /* execute the initialization code */
    execute(h_init);

    /* turn handling loop */
    for (;;) {

        /* execute the update code */
        execute(h_update);

        /* parse the next input command */
        if (parse()) {
            if (single())
                while (next() && single())
                    ;
        }

        /* parse error, call the error handling code */
        else
            execute(h_error);
    }
}

/* single - handle a single action */
int single()
{
    /* execute the before code */
    switch (execute(h_before)) {
    case ABORT: /* before handler aborted sequence */
        return (FALSE);
    case CHAIN: /* execute the action handler */
        if (execute(getafield(getvalue(V_ACTION),A_CODE)) == ABORT)
            return (FALSE);
    case FINISH:/* execute the after code */
        if (execute(h_after) == ABORT)
            return (FALSE);
        break;
    }
    return (TRUE);
}
```

continued

```
/* error - print an error message and exit */
error(msg)
  char *msg;
{
    trm_str(msg);
    trm_chr('\n');
    exit();
}
```

ADVJUNK.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advjunk.c - operating system specific code for "advint" */

#include <stdio.h>

long _seed = 1L;

int rand()
{
    _seed *= 397204094L;
    return (_seed & 0x7FFF);}

srand(n)
  long n;
{
   _seed = n;
}

int getch()
{
    int ch;
    if ((ch = bdos(1) & 0xFF) == '\r') { bdos(6,'\n'); ch = '\n'; }
    return (ch);
}

waitch()
{
    bdos(7);
}

putch(ch,fp)
  int ch; FILE *fp;
{
    aputc(ch,fp);
}

int advsave(hdr,hlen,save,slen)
  char *hdr; int hlen; char *save; int slen;
{
    char fname[50];
    int fd;

    trm_str("File name? ");
    trm_get(fname);

    /* add the extension */
    strcat(fname,".sav");

    /* create the data file */
    if ((fd = creat(fname,0666)) == -1)
        return (0);

    /* write the header */
    if (write(fd,hdr,hlen) != hlen) {
        close(fd);
        return (0);
    }
```

```
    /* write the data */
    if (write(fd,save,slen) != slen) {
        close(fd);
        return (0);
    }

    /* close the file and return successfully */
    close(fd);
    return (1);
}

int advrestore(hdr,hlen,save,slen)
  char *hdr; int hlen; char *save; int slen;
{
    char fname[50],hbuf[50],*p;
    int fd;

    if (hlen > 50)
        error("save file header buffer too small");

    trm_str("File name? ");
    trm_get(fname);

    /* add the extension */
    strcat(fname,".sav");

    /* create the data file */
    if ((fd = open(fname,0)) == -1)
        return (0);

    /* read the header */
    if (read(fd,hbuf,hlen) != hlen) {
        close(fd);
        return (0);
    }

    /* compare the headers */
    for (p = hbuf; hlen--; )
        if (*hdr++ != *p++) {
            trm_str("This save file does not match the adventure!\n");
            return (0);
        }

    /* read the data */
    if (read(fd,save,slen) != slen) {
        close(fd);
        return (0);
    }

    /* close the file and return successfully */
    close(fd);
    return (1);
}
```

ADVEXE.C Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
/* advexe.c - adventure code executer */
/*
        Copyright (c) 1986, by David Michael Betz
        All rights reserved
*/

#include "advint.h"
#include "advdbs.h"

/* external variables */
```

continued

```
extern char line[];
extern int nouns[],*adjectives[];

/* local variables */
int pc,opcode,p2,p3,sts;
int stack[STKSIZE],*sp,*fp,*top;
long rseed = 1L;

/* external routines */
extern long time();

/* execute - execute adventure code */
int execute(code)
  int code;
{
    /* setup initial program counter */
    if ((pc = code) == NIL)
        return (CHAIN);

    /* initialize */
    sp = fp = top = stack + STKSIZE;

    /* execute the code */
    for (sts = 0; sts == 0; )
        exe_one();

    return (sts);
}

/* exe_one - execute one instruction */
exe_one()
{
    /* get the opcode */
    opcode = getcbyte(pc); pc++;

    /* execute the instruction */
    switch (opcode) {
    case OP_CALL:
                *--sp = getboperand();
                *--sp = pc;
                *--sp = (int)(top - fp);
                fp = sp;
                pc = getafield(fp[fp[2]+3],A_CODE);
                break;
    case OP_SEND:                   *--sp = getboperand();
                *--sp = pc;
                *--sp = (int)(top - fp);
                fp = sp;
                if (p2 = fp[fp[2]+3])
                    p2 = getofield(p2,O_CLASS);
                else
                    p2 = fp[fp[2]+2];
                if (p2 && (p2 = getp(p2,fp[fp[2]+1]))) {
                    pc = getafield(p2,A_CODE);
                    break;
                }
                *sp = NIL;
                /* return NIL if there is no method for this message */
    case OP_RETURN:
                if (fp == top)
                    sts = CHAIN;
                else {
                    p2 = *sp;
                    sp = fp;
                    fp = top - *sp++;
                    pc = *sp++;
                    p3 = *sp++;
                    sp += p3;
                    *sp = p2;
                }
                break;
    case OP_TSPACE:
                sp -= getboperand();
                break;
    case OP_TMP:
                p2 = getboperand();
```

```
            *sp = fp[-p2-1];
            break;
case OP_TSET:
            p2 = getboperand();
            fp[-p2-1] = *sp;
            break;
case OP_ARG:
            p2 = getboperand();
            if (p2 >= fp[2])
                error("too few arguments");
            *sp = fp[p2+3];
            break;
case OP_ASET:
            p2 = getboperand();
            if (p2 >= fp[2])
                error("too few arguments");
            fp[p2+3] = *sp;
            break;
case OP_BRT:
            pc = (*sp ? getwoperand() : pc+2);
            break;
case OP_BRF:
            pc = (*sp ? pc+2 : getwoperand());
            break;
case OP_BR:
            pc = getwoperand();
            break;
case OP_T:
            *sp = T;
            break;
case OP_NIL:
            *sp = NIL;
            break;
case OP_PUSH:
            *--sp = NIL;
            break;
case OP_NOT:
            *sp = (*sp ? NIL : T);
            break;
case OP_ADD:
            p2 = *sp++;
            *sp += p2;
            break;
case OP_SUB:
            p2 = *sp++;
            *sp -= p2;
            break;
case OP_MUL:
            p2 = *sp++;
            *sp *= p2;
            break;
case OP_DIV:
            p2 = *sp++;
            *sp = (p2 == 0 ? 0 : *sp / p2);
            break;
case OP_REM:
            p2 = *sp++;
            *sp = (p2 == 0 ? 0 : *sp % p2);
            break;
case OP_BAND:
            p2 = *sp++;
            *sp &= p2;
            break;
case OP_BOR:
            p2 = *sp++;
            *sp |= p2;
            break;
case OP_BNOT:
            *sp = *sp;
            break;
case OP_LT:
            p2 = *sp++;
            *sp = (*sp < p2 ? T : NIL);
            break;
```

continued

```
    case OP_EQ:
                p2 = *sp++;
                *sp = (*sp == p2 ? T : NIL);
                break;    case OP_GT:
                p2 = *sp++;
                *sp = (*sp > p2 ? T : NIL);
                break;
    case OP_LIT:
                *sp = getwoperand();
                break;
    case OP_SPLIT:
                *sp = getboperand();
                break;
    case OP_SNLIT:
                *sp = -getboperand();
                break;
    case OP_VAR:
                *sp = getvalue(getwoperand());
                break;
    case OP_SVAR:
                *sp = getvalue(getboperand());
                break;
    case OP_SET:
                setvalue(getwoperand(),*sp);
                break;
    case OP_SSET:
                setvalue(getboperand(),*sp);
                break;
    case OP_GETP:
                p2 = *sp++;
                *sp = getp(*sp,p2);
                break;
    case OP_SETP:
                p3 = *sp++;
                p2 = *sp++;
                *sp = setp(*sp,p2,p3);
                break;
    case OP_PRINT:
                print(*sp);
                break;
    case OP_PNUMBER:
                pnumber(*sp);
                break;
    case OP_PNOUN:
                show_noun(*sp);
                break;
    case OP_TERPRI:
                trm_chr('\n');
                break;
    case OP_FINISH:
                sts = FINISH;
                break;
    case OP_CHAIN:
                sts = CHAIN;
                break;
    case OP_ABORT:
                sts = ABORT;
                break;    case OP_EXIT:
#ifdef MAC
                macpause();
#endif
                trm_done();
                exit();
                break;
    case OP_YORN:
                trm_get(line);
                *sp = (line[0] == 'Y' || line[0] == 'y' ? T : NIL);
                break;
    case OP_CLASS:
                *sp = getofield(*sp,O_CLASS);
                break;
    case OP_MATCH:
                p2 = *sp++;
                *sp = (match(*sp,nouns[p2-1],adjectives[p2-1]) ? T : NIL);
                break;
```

```
    case OP_SAVE:
                *sp = db_save();
                break;
    case OP_RESTORE:
                *sp = db_restore();
                break;
    case OP_RESTART:
                *sp = db_restart();
                break;
    case OP_RAND:
                *sp = getrand(*sp);
                break;
    case OP_RNDMIZE:
                setrand(time(0L));
                *sp = NIL;
                break;
    default:
            if (opcode >= OP_XVAR && opcode < OP_XSET)
                *sp = getvalue(opcode - OP_XVAR);
            else if (opcode >= OP_XSET && opcode < OP_XPLIT)
                setvalue(opcode - OP_XSET,*sp);
            else if (opcode >= OP_XPLIT && opcode < OP_XNLIT)
                *sp = opcode - OP_XPLIT;
            else if (opcode >= OP_XNLIT && opcode < 256)
                *sp = OP_XNLIT - opcode;
            else
                trm_str("Bad opcode\n");
            break;
    }
}

/* getboperand - get data byte */
int getboperand()
{
    int data;
    data = getcbyte(pc); pc += 1;
    return (data);}

/* getwoperand - get data word */
int getwoperand()
{
    int data;
    data = getcword(pc); pc += 2;
    return (data);
}

/* print - print a message */
print(msg)
  int msg;
{
    int ch;

    msg_open(msg);
    while (ch = msg_byte())
        trm_chr(ch);
}

/* pnumber - print a number */
pnumber(n)
  int n;
{
    char buf[10];

    sprintf(buf,"%d",n);
    trm_str(buf);
}

/* getrand - get a random number between 0 and n-1 */
int getrand(n)
  int n;
{
    long k1;

    /* make sure we don't get stuck at zero */
    if (rseed == 0L) rseed = 1L;
```

continued

```
    /* algorithm taken from Dr. Dobbs Journal, November 1985, page 91 */
    k1 = rseed / 127773L;
    if ((rseed = 16807L * (rseed - k1 * 127773L) - k1 * 2836L) < 0L)
        rseed += 2147483647L;

    /* return a random number between 0 and n-1 */
    return ((int)(rseed % (long)n));
}

/* setrand - set the random number seed */
setrand(n)
  long n;
{
    rseed = n;
}
```

ADVSYS.DOC Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

ADVSYS - An Adventure Writing System

Version 1.2

by David Betz
127 Taylor Road
Peterborough, NH 03458
(603) 924-6936 (home)

July 14, 1986

INTRODUCTION

ADVSYS is a special purpose programming language that was specifically designed to be used to write computer text adventure games. It includes a facility for defining the kinds of objects that are common in adventures. Some objects represent locations on the game map, some objects represent things that the player can find while exploring the adventure world, and some objects represent other characters that the adventurer can encounter during his or her journeys. The adventure language also provides a facility to define actions. Actions are short sections of code that determine what happens in response to a command from the player. These two concepts, "objects" and "actions" form the basis for the adventure language.

ACKNOWLEDGEMENTS

Although I have written all of the code associated with this adventure writing system, I must acknowledge the assistance of one individual without whom this project would probably never have reached completion. That person is Gary McGath. Gary was interested in writing a commercial quality adventure game and I convinced him to write it using my system (which was as yet almost completely unspecified) instead of using a traditional programming language. The input that Gary provided during the development of his game contributed significantly to the overall design of the system. I would like to thank Gary for that contribution.

USING THE SYSTEM TO WRITE AN ADVENTURE

In order to write an adventure using this system, you need to write an adventure description. This is an ordinary ASCII text file containing definitions for all of the objects and actions in your adventure game. This file is used as input to the adventure compiler. The compiler takes the adventure description and compiles it into a set of data structures.

In order to play an adventure written using this system, you need the data structure file that was produced by the compiler and the adventure interpreter program. The interpreter uses the information produced by the adventure compiler to allow a player to play the adventure game. Notice that it is not necessary for the player to have access to the original adventure description file. All of the information that is necessary to play the adventure game is contained within the data structure file that is produced by the compiler. This file is a binary file that cannot be simply "listed" to reveal the internal workings of the adventure.

The adventure compiler is called ADVCOM and the interpreter is called ADVINT. These two programs in conjunction with this documentation are all that is required to write and play adventure games using this system.

RUNNING THE COMPILER

If you have created an adventure definition file called "MYADV.ADV", you can compile it with the command:

```
A>advcom myadv
```

Typing this command will invoke the adventure compiler and cause it to compile the file named "MYADV.ADV". The ".ADV" extension is added to the file name by the compiler. During the process of compiling the file, many messages will be printed telling about the progress of the compiler. At the end of the compilation process, the compiler prints a set of statistics describing the resulting data structure file. This file will be called "MYADV.DAT". It contains the data structures needed by the adventure interpreter to allow a player to play the adventure game.

Note: The "A>" in the line above is the MS-DOS prompt and should not be typed as part of the command.

RUNNING THE INTERPRETER

Assuming that you have a compiled adventure data file called "MYADV.DAT", you can play the adventure by typing the command:

```
A>advint myadv
```

This command will start the adventure. There will probably be some text printed at this point describing the adventure and the initial situation. You will then be prompted to type a command. The prompt is the colon character. The format for commands is described under the section about the parser. After typing a command, you will be told what happened as a result of your command, your new situation will be described and you will begin the loop again.

continued

ADVENTURE DESCRIPTION FILE FORMAT

All adventure description files contain a collection of statements. These statements must be formed according to the following rules:

The adventure definition statement:

All adventure definitions should have an ADVENTURE statement. This statement gives the name of the adventure and the version number of the definition file. Each adventure should have a unique name. This name is used to identify "saved position" files and insure that only files that correspond to the current adventure are restored. The version number allows the author to have many versions of the same adventure during development and guarantee that "save" files from one version aren't restored into another version.

```
(ADVENTURE name version)
```

Example:

```
(ADVENTURE sample 1)
```

Vocabulary statements:
These statements add words to the adventure vocabulary.

```
(ADJECTIVE word*)
(PREPOSITION word*)
(CONJUNCTION word*)
(ARTICLE word*)
(SYNONYM word synonym*)
```

Examples:

```
(ADJECTIVE red blue)
(CONJUNCTION and)
(SYNONYM big large)
```

Note:

Words are also added to the vocabulary by the object and action definitions using the NOUN, ADJECTIVE, VERB and PREPOSITION statements.

Constant definition statement:

```
(DEFINE name value)
```

Examples:

```
(DEFINE what "I don't understand what you're saying!\n")
(DEFINE max-load 100)
```

Function definition statement:

```
(DEFINE (function-name [arg-name]* [&aux tmp-name*]) expr*)
```

Example:

```
(DEFINE (factorial n)
  (IF (< n 2)
    1
    (* n (factorial (- n 1)))))
```

Variable definition statement:

```
(VARIABLE variable-name*)
```

Example:

```
(VARIABLE score i j)
```

Property name definition statement:

```
(PROPERTY property-name*)
```

Example:

```
(PROPERTY weight value)
```


Comments:

Comments begin with a semi-colon and end with the end of the line.

Example:

```
; this is a comment
```

Include files:

Any line that begins with a "@" causes the inclusion of another file. The file name immediately follows the at-sign and extends to the end of the line. Only one level of include is supported.

Example:

```
@basic.adv
```

continued

Handler definition statements:

```
(INIT expr*)
(UPDATE expr*)
(BEFORE expr*)
(AFTER expr*)
(ERROR expr*)
```

Example:

```
(INIT
  (print "Welcome to the sample adventure!\n"))
```

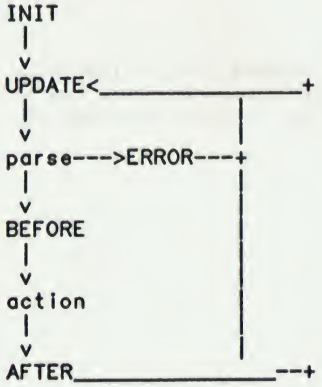
Handlers:

All activity within an adventure game is controlled by a built-in handler loop. Each of the handlers in the loop contains code that is provided by the adventure author. The sequencing from handler to handler is provided by the adventure system itself.

The first handler that is called in an adventure game is the INIT handler. It prints some sort of introductory text and initializes all global variables in order to start the adventure game.

After the INIT handler has completed, the normal loop is entered. It starts with the UPDATE handler. The UPDATE handler prepares for the player's next turn. It should describe the player's location if it has changed since the last turn. After the UPDATE handler completes, the parser is called. It prompts the player for a command, parses the command, sets the built-in parser varaibles and exits. Then the BEFORE handler is called. It is called before the action associated with the command to allow the adventure author to inspect the parser variables before proceeding to the action itself. After the BEFORE handler completes, the action itself is called (or whatever action is stored in the built-in variable $ACTION when the BEFORE handler completes). When the action completes, the AFTER handler is called to give the author a chance to handle events that happen only at the end of a successful turn. The ERROR handler is called when the parser detects an error.

The handler loop:



The parser:

The parser handles all commands from the player. It prompts the player when it is ready for a new command. The prompt is the colon character. When the player has typed a command, the parser breaks the command into phrases. The parser recognizes the following command forms:

```
[actor,] verb
[actor,] verb dobjects
[actor,] verb dobjects preposition iobject
[actor,] verb iobject dobjects
```

Where:

```
actor            ==> a noun phrase
verb             ==> the verb phrase (1 or 2 words)
dobjects         ==> dobject [conjunction dobject]*
dobject          ==> a noun phrase
preposition      ==> a preposition
iobject          ==> a noun phrase
noun phrase      ==> [article] [adjective]* noun
```

Examples:

```
Look
Take the red magic sword
Take the red sword and the blue bottle
Give the troll the red sword
Give the red sword to the troll
Troll, give me the sword
```

Notes:

Square brackets enclose optional phrases. An asterisk indicates zero or more of the preceeding element.

The fourth form above is treated as if the player had typed:

```
[actor,] verb dobject "to" iobject
```

Once the parser has broken the command into phrases, it assigns each noun phrase a number. It stores the number of the actor noun phrase in the built-in variable $ACTOR. It stores the first direct object noun phrase number in the variable $DOBJECT. It stores the number of direct objects in the variable $NDOBJECTS. It stores the indirect object noun phrase number in the variable $IOBJECT. If any of the noun phrases is missing from the command, the corresponding variable is set to NIL. The parser saves the verb phrase and preposition to use when determining which action to use to handle the command.

continued

Action definition statement:

Actions are used to handle player commands. Each time the parser finishes parsing a new command, it uses the verb phrase and the preposition to locate an action to handle the command. Each action specifies a kind of template that must match the command in order for the action to be called. The template consists of the words used in the verb phrase and preposition and the existance of the actor, direct object and indirect object noun phrases. Once the parser finds an action that matches the command, it stores the action in the built-in variable $ACTION and exits.

```
(ACTION action-name astat*)
    astat:
        (ACTOR [flag])
        (VERB verb*)
        (DIRECT-OBJECT [flag])
        (PREPOSITION word*)
        (INDIRECT-OBJECT [flag])
    flag:
        REQUIRED  must have the corresponding np  .
        OPTIONAL  may have the corresponding np
        FORBIDDEN must not have the corresponding np

    verb:
        word
        (word word)
```

Example:

```
(ACTION take
  (VERB take (pick up))
  (DIRECT-OBJECT)
  (CODE
    (print "You can't take the ")
    (print-noun $dobject)
    (print "!\n")))
```

If the ACTOR, DIRECT-OBJECT or INDIRECT-OBJECT statements are left out entirely, the settings of the corresponding flags are taken from the action default definitions. If there is no action default definition, the value FORBIDDEN is assumed. If any of these statements is present, but no flag is specified, it is treated as if the flag REQUIRED was specified.

Action default definition statement:

This statement defines default values for the ACTOR, DIRECT-OBJECT and INDIRECT-OBJECT flags.

```
(DEFAULT dstat*)
    dstat:
        (ACTOR [flag])
        (DIRECT-OBJECT [flag])
        (INDIRECT-OBJECT [flag])
    flag:
        REQUIRED
        OPTIONAL
        FORBIDDEN
```

Example:

```
(DEFAULT
  (ACTOR OPTIONAL))
```

Object definition statements:

The object definition statements are used to define individual objects and classes of objects. The most basic way of defining an object is using the (OBJECT ...) statement. This defines an object which has no parent class.

It is also possible to create a class of objects that share information. A class is defined just like a normal object. It is given nouns, adjectives and properties. In addition, a class may have class properties. These are properties that are shared amongst all instances of the class. In order to create an instance of a class, the (class-name ...) form is used. This creates an instance of the named class. An instance will inherit all nouns and adjectives from its parent class. It will also inherit all class properties defined in the parent (and its parents). Any normal properties defined in the parent class will be copied to the new object. The copies will have the same values that the parent has, but it is possible for the instance to have property definitions that override these values. Instances may also have additional nouns, adjectives and properties.

```
(OBJECT object-name ostat*)
(class-name object-name ostat*)
    ostat:
        (NOUN word*)
        (ADJECTIVE word*)                          (PROPERTY [property-name value]*)
        (CLASS-PROPERTY [property-name value]*)
        (METHOD (selector [arg-name]* [&aux tmp-name*])
            expr*)
    class-name:
        the name of a previously defined object
```

Examples:

```
(OBJECT sword
  (NOUN sword weapon)
  (CLASS-PROPERTY
    is-weapon T)
  (PROPERTY
    weight 10
    value 5
    damage 20))

(sword red-sword
  (ADJECTIVE red)
  (PROPERTY
    damage 25))
```

continued

Expressions:

```
(+ expr expr)     add
(- expr expr)     subtract
(* expr expr)     multiply
(/ expr expr)     divide
(% expr expr)     remainder

(& expr expr)     bit-wise and
(| expr expr)     bit-wise or
( expr)          bit-wise complement
```

These arithmetic functions operate on integers. As it turns out, every data type in the system is represented by an integer, so these functions will work with any type of arguments. They are probably only useful with integers, however.

```
(RANDOMIZE)       reset the random number generator
(RAND expr)       generate a random number
```

These functions enable the generation of pseudo-random numbers. The (RAND n) function takes a single argument and generates a random number between zero and n-1. (RANDOMIZE) resets the seed used by the random number function so that each invocation of a program results in a new sequence of random numbers.

```
(AND expr*)       logical and (short circuits)
(OR expr*)        logical or (short circuits)          (NOT expr)        logical not
```

These functions operate on logical values. In this system, any value that is not equal to NIL (or zero) is considered true. NIL and zero are considered false. AND and OR evaluate their arguments from left to right and stop as soon as the value of the entire expression can be determined. In other words, AND stops when it encounters a false value, OR stops when it encounters a true value.

```
(< expr expr)     less than
(= expr expr)     equal to
(> expr expr)     greater than
```

These functions compare integers. They cannot be used to compare strings.

```
(GETP obj property-name)          get the value of a property
(SETP obj property-name value)    set the value of a property
```

These functions manipulate object properties. They are used to find the value of a property or to set the value of a property. They will also find and set the values of inherited properties. If GETP is used to find the value of a property that doesn't exist for the specified object, NIL is returned. If SETP is used to set the value of a property that doesn't exist, the operation is ignored.

```
(CLASS obj)
```

This function returns the class of an object. If the object was defined with an (OBJECT ...) statement, NIL will be returned. If the object was defined with the (class-name ...) statement, the class object will be returned.

```
(MATCH obj noun-phrase-number)
```

This function matches an object with a noun phrase. An object matches a noun phrase if it includes all of the adjectives specified in the noun phrase and also includes the noun mentioned. Both nouns and adjectives can be inherited.

```
(YES-OR-NO)        get a yes or no answer from the player
```

This function waits for the player to type a line. If the line begins with a 'Y' or a 'y', the function returns T. If the line begins with anything else, the function returns NIL.

```
(PRINT expr)                          print a string
(PRINT-NUMBER expr)                   print a number
(PRINT-NOUN noun-phrase-number)       print a noun phrase
(TERPRI)                              terminate the print line
```

These functions perform various sorts of output. PRINT prints strings, PRINT-NUMBER prints numbers and PRINT-NOUN prints a noun phrase.

```
(FINISH)          exit and continue with the AFTER handler
(CHAIN)           exit and continue with the next handler
(ABORT)           exit and continue with the UPDATE handler
(RESTART)         exit and restart the current game
(EXIT)            exit to the operating system
```

These functions cause the immediate termination of the current handler. FINISH causes execution to proceed with the AFTER handler, CHAIN causes execution to proceed with the next handler in the normal sequence, ABORT causes execution to proceed with the UPDATE handler (effectively aborting the current turn), RESTART restores the game to its

continued

original state and starts over with the INIT handler and EXIT causes an immediate exit back to the operating system.

```
(SAVE)          save the current game position
(RESTORE)       restore a saved game position
```

These functions allow the player to save and restore positions in the game. They prompt the player for a file name and either read a saved game position from the file or write the current game position to the file.

```
(function-name expr*)
```

This expression invokes a user defined function. There should be one expression for each of the formal arguments of the user function. The value of the expression is the value of the last expression in the body of the user function or the value passed to a RETURN statement within the function.

```
(SEND object selector [expr]*)
```

This expression sends a message to an object. The "object" expression should evaluate to an object. The selector should match a method selector for that object or one of its super-classes. The matching method is invoked with the specified expressions as arguments. Also, the implied argument SELF will refer to the object receiving the message.

```
(SEND-SUPER selector [expr]*)
```

This expression sends a message to the super-class of the current object. It can only be used within a method and it will cause the message to be passed up the class heirarchy to the super-class of the object refered to by SELF.

```
(SETQ variable value)
```

This expression sets the value of a user variable.

```
(COND [(test expr*)]*)                  execute conditionally
(IF test then-expr [else-expr])         traditional if-then-else
(WHILE test expr*)                      conditional iteration
(PROGN expr*)                           block construct
(RETURN [expr])                         return from a function
```

These statements are control constructs.

Primary expressions:

integer	(digits preceeded by an optional sign)
string	(characters enclosed in double quotes)
action-name	(an action name)
object-name	(an object or class name)
property-name	(a property name)
constant-name	(a defined constant or function)
variable-name	(a variable name)

Since an adventure description contains a large quantity of running text, the format for specifying string constants is somewhat extended from normal programming languages. In this system, a string is enclosed in double quotes. If the end of line occurs before the closing quote within a string, it is treated as if it were a space. Any number of consecutive spaces is collapsed into a single space. Also, the character pair "\n" is used to represent the "end of line" character, the pair "\t" is used to represent the tab character and the pair "\\" is used to represent the backslash character.

Examples:

```
"This is a string.\n"
"This
 is
 a
 string.\n"
```

Both of the examples above represent the same string.

Definitions of symbols used above:

expr	an expression
value	an expression
test	an expression (NIL means false, anything else is true)
then-expr	an expression
else-expr	an expression
obj	an expression that evaluates to an object
property-name	an expression that evaluates to a property name
noun-phrase-number	an expression that evaluates to a noun phrase number
variable	a variable name
T	true
NIL	false

Built-in variables set by the parser:

$ACTOR	(actor noun phrase number)
$ACTION	(action)
$DOBJECT	(first direct object noun phrase number)
$NDOBJECTS	(number of direct object noun phrases)
$IOBJECT	(indirect object noun phrase number)

Other built-in variables:

$OCOUNT	(total number of objects in the system)

CURRENT COMPILER LIMITS

500	words
500	objects
20	properties per object
200	actions or functions
16384	bytes of code
16384	bytes of data
262144	bytes of text

continued

OSAMPLE.ADV Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
; OSAMPLE.ADV
;
; This is a VERY simple sample adventure that uses the "OBJECTS.ADI"
; runtime support code.  It isn't interesting to play, but does illustrate
; most of the features of the adventure authoring system.  Try compiling
; it using the command:
;
;       A>ADVCOM OSAMPLE
;
; When the compile has finished, run the adventure using the command:
;
;       A>ADVINT OSAMPLE
;
; You should then see the initial welcome message and a description of
; your initial location.  You can use the direction names to move from
; one location to the next (or the abreviations N,S,E,W).  You should try
; manipulating objects using TAKE and DROP.  You can manipulate more than
; one at a time by using the conjunction AND.  You can also GIVE an object
; to another creature like the DOG or CAT.  You can instruct another
; creature to perform an action like:
;
;       CAT, GIVE THE DOG THE KEY
;
; You can also experiment with using adjectives to distinguish between
; objects (there is more than one KEY in this adventure).

(adventure sample 1)

(define welcome "Welcome to the sample adventure.\n")

@objects.adi

(actor adventurer
  (noun me)
  (property
    initial-location livingroom))

(actor dog
  (noun dog)
  (adjective small)
  (property
    description "There is a small dog here."
    short-description "a small dog"
    initial-location kitchen))

(actor cat
  (noun cat)
  (property
    description "There is a cat here."
    short-description "a cat"
    initial-location kitchen))

(location storage-room
  (property
    description "You are in a small storage room with many empty shelves.
                 The only exit is a door to the west."
    short-description "You are in the storage room."
    west hallway)  (method (leave obj dir)
    (if (send obj carrying? rusty-key)
      (send-super leave obj dir)
      (print "You seem to be missing something!\n"))))

(location hallway
  (property
    description "You are in a long narrow hallway. There is a door to the
                 east into a small dark room.  There are also exits on both
                 the north and south ends of the hall."
    short-description "You are in the hallway."
```

```
    east storage-room
    north kitchen
    south livingroom))

(location kitchen
  (property
    description "This is a rather dusty kitchen.  There is a hallway to the
                 south and a pantry to the west."
    short-description "You are in the kitchen."
    south hallway
    west pantry))

(location pantry
  (property
    description "This is the kitchen pantry.  The kitchen is through a
                 doorway to the east."
    short-description "You are in the pantry."
    east kitchen))

(location livingroom
  (property
    description "This appears to be the livingroom.  There is a hallway to
                 the north and a closet to the west."
    short-description "You are in the livingroom."
    north hallway
    west closet
    south front-door-1))

(location outside
  (property
    description "You are outside a small house.  The front door is to the
                 north."
    short-description "You are outside."
    north front-door-2))

(portal front-door
  (noun door)
  (adjective front)
  (class-property
    short-description "front door"
    closed t
    locked t
    key rusty-key))
(front-door front-door-1
  (property
    initial-location livingroom
    other-side front-door-2))

(front-door front-door-2
  (property
    initial-location outside
    other-side front-door-1))

(location closet
  (property
    description "This is the livingroom closet.  The livingroom is through
                 a doorway to the east."
    short-description "You are in the closet."
    east livingroom))

(thing rusty-key
  (noun key)
  (adjective rusty)
  (property
    description "There is a rusty key here."
    short-description "a rusty key"
    initial-location storage-room))

(thing silver-key
  (noun key)
  (adjective small silver)
  (property
    description "There is a small silver key here."
    short-description "a small silver key"
    initial-location closet))
```

continued

OBJECTS.ADI Contributed by: David Betz
TEXT "An Adventure Authoring System," David Betz, May 1987, page 125.

```
; This is the object-oriented runtime package
; by David Betz
; July 19, 1986

; ********************
; PROPERTY DEFINITIONS
; ********************

; These properties will be used for connections between locations
(property
  north                 ; the location to the north
  south                 ; the location to the south
  east                  ; the location to the east  west                ; the location to the west
  up                    ; the location above
  down)                 ; the location below

; Basic object properties
(property
  initial-location      ; the initial location of a "thing"
  description           ; the "long" description of a location
  short-description)    ; the "short" description of a location

; Connection properties
(property
  parent                ; the parent of an object
  sibling               ; the next sibling of an object
  child)                ; the first child of an object

; Location properties
(property
  visited)              ; true if location has been visited by player

; Portal properties
(property
  closed                ; true if the portal is closed
  locked                ; true if the portal is locked
  key                   ; key to unlock the portal
  other-side)           ; the other portal in a pair

; ***********************
; VOCABULARY DEFINITIONS
; ***********************

; Some abbreviations for common commands
(synonym north n)
(synonym south s)
(synonym east e)
(synonym west w)
(synonym inventory i)

; Define the basic vocabulary
(conjunction and)
(article the that)

; ********************
; VARIABLE DEFINITIONS
; ********************

(variable
  curloc                ; the location of the player character
  %actor                ; the actor object
  %dobject              ; the direct object
  %iobject)             ; the indirect object

; *********************
; CONNECTION PRIMITIVES
; *********************
; Connect an object to a parent
(define (connect p c)
```

```
  (setp c parent p)
  (setp c sibling (getp p child))
  (setp p child c))

; Connect all objects to their initial parents
(define (connect-all &aux obj maxp1 par)
  (setq obj 1)
  (setq maxp1 (+ $ocount 1))
  (while (< obj maxp1)
    (if (setq par (getp obj initial-location))
      (connect par obj))
    (setq obj (+ obj 1))))

; Disconnect an object from its current parent
(define (disconnect obj &aux this prev)
  (setq this (getp (getp obj parent) child))
  (setq prev nil)
  (while this
    (if (= this obj)
      (progn
        (if prev
          (setp prev sibling (getp this sibling))
          (setp (getp this parent) child (getp this sibling)))
        (setp this parent nil)
        (return)))
    (setq prev this)
    (setq this (getp this sibling))))

; Print the contents of an object (used by "look")
(define (print-contents obj prop &aux desc)
  (setq obj (getp obj child))
  (while obj
    (if (setq desc (getp obj prop))
      (progn
        (print " ")
        (print desc)))
    (setq obj (getp obj sibling))))

; List the contents of an object (used for "inventory")
(define (list-contents obj prop &aux desc)
  (setq obj (getp obj child))
  (while obj
    (if (setq desc (getp obj prop))
      (progn
        (print "\t")
        (print desc)
        (terpri)))
    (setq obj (getp obj sibling))))

; ************************
; OBJECT CLASS DEFINITIONS
; ************************
; ************************
; The "basic-thing" class
; ************************

(object basic-thing
  (property
    parent nil          ; the parent of this object
    sibling nil))       ; the next sibling of this object

; ***************************
; The "location" object class
; ***************************

(object location
  (property
    child nil           ; the first object in this location
    visited nil)        ; has the player been here yet?
  (method (knock? obj)
    T)
  (method (enter obj)
    (connect self obj)
    T)
```

continued

```
(method (leave obj dir &aux loc)
  (if (setq loc (getp self dir))
    (if (send loc knock? obj)
      (progn
        (disconnect obj)
        (send loc enter obj)))
    (progn
      (print "There is no exit in that direction.\n")
      nil)))
(method (describe)
  (if (getp self visited)
    (print (getp self short-description))
    (progn
      (print (getp self description))
      (print-contents self description)
      (setp self visited t)))
  (terpri)))

; ******************
; The "portal" class
; ******************

(basic-thing portal
  (method (knock? obj)
    (if (getp self closed)
      (progn
        (print "The ")
        (print (getp self short-description))
        (print " is closed!\n")
        nil)
      T))  (method (enter obj)
    (connect (getp (getp self other-side) parent) obj)))
  (method (open)
    (if (not (getp self closed))
      (progn
        (print "The ")
        (print (getp self short-description))
        (print " is already open!\n")
        nil)
      (if (getp self locked)
        (progn
          (print "The ")
          (print (getp self short-description))
          (print " is locked!\n")
          nil)
        (progn
          (setp self closed nil)
          T))))
  (method (close)
    (if (getp self closed)
      (progn
        (print "The ")
        (print (getp self short-description))
        (print " is already closed!\n")
        nil)
      (progn
        (setp self closed T)
        T)))
  (method (lock thekey)
    (if (not (getp self closed))
      (progn
        (print "The ")
        (print (getp self short-description))
        (print " is not closed!\n")
        nil)
      (if (getp self locked)
        (progn
          (print "The ")
          (print (getp self short-description))
          (print " is already locked!\n")
          nil)
        (if (not (= thekey (getp self key)))
          (progn
            (print "It doesn't fit the lock!\n")
            nil)
          (progn
```

```
            (setp self locked t)
            T)))))
   (method (unlock thekey)
     (if (not (getp self closed))
       (progn
         (print "The ")
         (print (getp self short-description))
         (print " is already open!\n")
         nil)      (if (not (getp self locked))
         (progn
           (print "The ")
           (print (getp self short-description))
           (print " is not locked!\n")
           nil)
         (if (not (= thekey (getp self key)))
           (progn
             (print "It doesn't fit the lock!\n")
             nil)
           (progn
             (setp self locked nil)
             T))))))

; *****************
; The "actor" class
; *****************

(basic-thing actor
  (property
    child nil)          ; the first "thing" carried by this actor
  (method (move dir)
    (send (getp self parent) leave self dir))
  (method (take obj)
    (disconnect obj)
    (connect self obj))
  (method (drop obj)
    (disconnect obj)
    (connect (getp self parent) obj))
  (method (carrying? obj)
    (= (getp obj parent) self))
  (method (inventory)
    (cond ((getp %actor child)
           (print "You are carrying:\n")
           (list-contents %actor short-description))
          (T (print "You are empty-handed.\n")))))

; *****************
; The "thing" class (things that can be taken)
; *****************

(basic-thing thing
  (class-property
    takeable t))

; ****************************
; The "stationary-thing" class (things that can't be moved)
; ****************************

(basic-thing stationary-thing)

; ***********************
; MISCELLANEOUS FUNCTIONS
; ***********************
; Complain about a noun phrase
(define (complain head n tail)
  (print head)
  (print-noun n)
  (print tail)
  (abort))

; Find an object in a location
(define (findobject loc n &aux this found)
  (setq this (getp loc child))
  (setq found nil)
  (while this
    (if (match this n)
```

continued

```
      (if found
        (complain "I don't know which " n " you mean!\n")
        (setq found this)))
    (setq this (getp this sibling)))
  found)

; Find an object in the player's current location
;  (or in the player's inventory)
(define (in-location n &aux obj)
  (if (or (setq obj (findobject curloc n))
          (setq obj (findobject %actor n)))
    obj
    (complain "I don't see a " n " here!\n")))

; Find an object in the player's inventory
;  (or in the player's current location)
(define (in-pocket n &aux obj)
  (if (or (setq obj (findobject %actor n))
          (setq obj (findobject curloc n)))
    obj
    (complain "You don't have a " n "!\n")))

; ***************
; ACTION DEFAULTS
; ***************

(default
  (actor optional))

; ******************
; ACTION DEFINITIONS
; ******************

(action look
  (verb look)
  (code
    (setp curloc visited nil)
    (send curloc describe)))

(action a-take
  (verb take get (pick up))
  (direct-object)  (code
    (setq %dobject (in-location $dobject))
    (if (getp %dobject takeable)
      (progn
        (if (send %actor carrying? %dobject)
          (complain "You are already carrying the " $dobject "!\n"))
        (send %actor take %dobject)
        (print-noun $dobject)
        (print " taken.\n"))
      (complain "You can't take the " $dobject "!\n"))))

(action take-err
  (verb take get (pick up))
  (code
    (print "Take what?\n")))

(action a-drop
  (verb drop (put down))
  (direct-object)
  (code
    (setq %dobject (in-pocket $dobject))
    (if (send %actor carrying? %dobject)
      (progn
        (send %actor drop %dobject)
        (print-noun $dobject)
        (print " dropped.\n"))
      (complain "You aren't carrying the " $dobject "!\n"))))

(action drop-err
  (verb drop (put down))
  (code
    (print "Drop what?\n")))

(action give
  (verb give)
```

```
    (direct-object)
    (preposition to)
    (indirect-object)
    (code
      (setq %dobject (in-pocket $dobject))
      (setq %iobject (in-location $iobject))
      (if (send %actor carrying? %dobject)
        (progn
          (send %actor drop %dobject)
          (send %iobject take %dobject)
          (print-noun $dobject)
          (print " given.\n"))
        (complain "You aren't carrying the " $dobject "!\n"))))

  (action give-err
    (verb give)
    (direct-object optional)
    (code
      (if $dobject
        (complain "Give the " $dobject " to whom?\n"))
        (print "Give what?\n")))

  (action a-inventory
    (verb inventory)
    (code
      (send %actor inventory)))

  ; ***************
  ; PORTAL COMMANDS
  ; ***************

  (action a-open
    (verb open)
    (direct-object)
    (code
      (setq %dobject (in-location $dobject))
      (send %dobject open)))

  (action open-err
    (verb open)
    (code
      (print "Open what?\n")))

  (action a-close
    (verb close)
    (direct-object)
    (code
      (setq %dobject (in-location $dobject))
      (send %dobject close)))

  (action close-err
    (verb close)
    (code
      (print "Close what?\n")))

  (action a-lock
    (verb lock)
    (direct-object)
    (preposition with)
    (indirect-object)
    (code
      (setq %dobject (in-location $dobject))
      (setq %iobject (in-pocket $iobject))
      (send %dobject lock %iobject)))

  (action lock-err
    (verb lock)
    (direct-object optional)
    (code
      (if $dobject
        (complain "Lock the " $dobject " with what?\n"))
        (print "Lock what?\n")))
```

continued

```
(action a-unlock
  (verb unlock)  (direct-object)
  (preposition with)
  (indirect-object)
  (code
    (setq %dobject (in-location $dobject))
    (setq %iobject (in-pocket $iobject))
    (send %dobject unlock %iobject)))

(action unlock-err
  (verb unlock)
  (direct-object optional)
  (code
    (if $dobject
      (complain "Unlock the " $dobject " with what?\n"))
      (print "Unlock what?\n")))

; *********************
; GAME CONTROL COMMANDS
; *********************

(action save
  (verb save)
  (code
    (save)))

(action restore
  (verb restore)
  (code
    (restore)))

(action restart
  (verb restart)
  (code
    (restart)))

(action quit
  (verb quit)
  (code
    (print "Are you sure you want to quit? ")
    (if (yes-or-no)
      (exit))))

; **************
; TRAVEL ACTIONS
; **************

(action go-north
  (verb north (go north))
  (code
    (send %actor move north)))
(action go-south
  (verb south (go south))
  (code
    (send %actor move south)))
(action go-east
  (verb east (go east))
  (code
    (send %actor move east)))

(action go-west
  (verb west (go west))
  (code
    (send %actor move west)))

(action go-up
  (verb up (go up))
  (code
    (send %actor move up)))

(action go-down
  (verb down (go down))
  (code
    (send %actor move down)))
```

```
; ******************
; HANDLER DEFINITIONS
; ******************

(init
  (connect-all)
  (print welcome)
  (setq curloc nil))

(update
  (if (not (= (getp adventurer parent) curloc))
    (progn
      (setq curloc (getp adventurer parent))
      (send curloc describe))))

(before
  (setq %actor adventurer)
  (if $actor
    (progn
      (setq %actor (in-location $actor))
      (if (not (= (class %actor) actor))
        (complain "You can't talk to a " $actor "!\n")))))
```

LISTINGS.DOC Contributed by: Chris Crawford and Eva White
TEXT The four benchmarks that run on Macintosh SCSI external
hard disks as reviewed by Chris Crawford and Eva White, May 1987, page 237.

Listings for the four benchmarks run on Macintosh SCSI external
hardisks review by Chris Crawford and Eva White.

```
10 REM LineFileWrite
100 A$="12345678123456781234567812345678"
120 B$=A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$
160 OPEN "TEST" FOR OUTPUT AS #1
170 START=TIMER
180 FOR I=1 TO 1024
200 PRINT #1, B$
220 NEXT I
230 CLOSE
250 PRINT TIMER-START
260 PRINT "DONE"

300 REM LineFileRead
320 OPEN"TEST" FOR INPUT AS #1
330 START=TIMER
340 FOR I=1 TO 1024
360 B$=INPUT$(512,1)
380 NEXT I
400 CLOSE
410 PRINT TIMER-START
420 PRINT "DONE"

10 REM FragFileWrite
100 A$="12345678123456781234567812345678"
110 B$=A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$+A$
170 START=TIMER
175 FOR J=1 TO 64
180 FOR I=1 TO 16
185 C$="TEST"+STR$(I)
190 OPEN C$ FOR APPEND AS #1
200 PRINT #1, B$
210 CLOSE #1
220 NEXT I
230 NEXT J
250 PRINT TIMER-START
260 PRINT "DONE"

300 REM FragFileRead
330 START=TIMER
```

continued

```
335 FOR I=1 TO 16
340 C$="TEST"+STR$(I)
350 OPEN C$ FOR INPUT AS #1
360 B$=INPUT$(32767,1)
385 CLOSE #1
390 NEXT I410 PRINT TIMER-START
420 PRINT "DONE"
```

PULLDOWN.C Contributed by: James L. Pinson
TEXT "Pull-Down Menus in C," by James L. Pinson, May 1987, page 109.

```
/*   COPYRIGHT (C) 1986 */
/*   BY JAMES L PINSON  */
/*  ALL RIGHTS RESERVED */


/* Compiled with Lattice C V2.14 */
/* Computer: IBM PC JR           */
/* Text editor: Sidekick         */
/* Last revision 3/16/1987       */


#include  "stdio.h"
#include  "ctype.h"

#define void int


#define BLACK   0         /*  THESE ARE FOR COLOR CARDS */
#define BLUE    1
#define GREEN   2
#define CYAN    3
#define RED     4
#define MAGENTA 5
#define BROWN   6
#define WHITE   7
#define L_BLUE  9      /* LIGHT-BLUE FOREGROUND ONLY */
#define L_GREEN 10     /* LIGHT-GREEN FOREGROUND ONLY */
#define YELLOW  14
#define IWHITE  15     /* INTENSE-WHITE FOREGROUND ONLY*/


#define UNDERLINE 1      /* THESE ARE FOR MONOCHROME CARDS */
#define NORMAL    7
#define HI_INTEN  15
#define REVERSE   112

#define TRUE  1
#define FALSE 0

unsigned int page;           /* extern decl. for functions*/
unsigned int attribute;
unsigned int mon_type;
char wrt_meth= 'f';


#define  NU_MAIN 5         /* number of main menu options */
#define  NU_SUB  5         /* number of sub menu options  */

 struct menu_str{         /* change this if you need more options */
    char *head;
    char *body[NU_SUB];
    void (*fun1)();
    void (*fun2)();
    void (*fun3)();
```

```
    void (*fun4)();
    void (*fun5)();
  } ;


main(argc,argv)
     int argc;
     char **argv;
{
extern unsigned int page;
extern unsigned int attribute;
extern unsigned int mon_type;

char ch,ext;

int i,hi_attr,nor_attr;

int demo();
int help();
static  struct menu_str m_menu [NU_MAIN]={

    "   File    ",   /*  The first menu option       */
      "   Dir       ",   /*  Menu sub options */
      "   Load      ",
      "   Save      ",
      "   dElete    ",
      "   Path      ",
            demo,   /* The functions each sub-option call */
            demo,
            demo,    /* these all call the same fake function */
            demo,
            demo,

      "   fiNd    ",           /* The second menu option */
        " All-words  ",
        " First-word ",
        "\0",
        "\0",           /* space filler for unused option names */
        "\0",
            demo,
            demo,
            0,       /* unused function pointer */
            0,
            0,
      "   Configure    ",    /* The third option */
        " Modem ",
        " Screen ",
        " Printer ",
        "\0",
        "\0",
            demo,
            demo,
            demo,
            0,
            0,

      "        Output        ",          /* The fourth menu option */
        "  Screen  ",
        "  Printer ",
        "  Disk    ",
        "  Modem   ",
        "\0",
            demo,
            demo,
            demo,
            demo,
            0,


        " Help ",                       /* The fifth option */
        " Instant help (really works) ",
        "\0",
        "\0",
        "\0",
        "\0",
```

continued

```
                help,
                0,
                0,
                0,
                0,
};


/* was a slow write requested? */
if (tolower (*argv[1])== 's') wrt_meth = 's';

page=0;


 mon_type =(what_mon());

  if (mon_type==1){                          /* FIND OUT IF YOU HAVE A COLOR CARD
*/
    hi_attr=  set_color(BLACK,CYAN);         /* AND SET ATTRIBUTES ACCORDINGLY */
    nor_attr= set_color(WHITE,BLACK);
  }
    else{
     hi_attr = REVERSE;
     nor_attr = NORMAL;     }

 attribute = nor_attr;
 cursor(0);                  /* hide cursor */
 win_save('s');
 cls();

 if (mon_type==1) make_help();

 make_inst();            /* SHOW INSTRUCTIONS */

 menu(m_menu,NU_MAIN,NU_SUB,hi_attr,nor_attr);

 win_save('r');          /* restore text display*/
 cursor(1);              /* restore cursor      */
}


int menu (m_menu,nu_main,nu_sub,hi_attr,nor_attr)
struct menu_str m_menu[];
int nu_main,nu_sub,hi_attr, nor_attr;

{
extern unsigned int page;
extern unsigned int attribute;
extern unsigned int mon_type;

int i,j,k,cur_x,cur_y,cur_opt,found,expert=1;
char ch, ext, itr;

   ch= ' ';ext=' '; cur_opt=0; found =0;

if (mon_type==1) attribute = set_color(YELLOW,BLACK);
else attribute = nor_attr;

make_window(1,1,78,1,1);


  for(;;){                 /* endless loop */

        for(i=0;i<nu_main;i++){
          j=0;
          while(itr = m_menu[i].head[j++]){
            if ( ch==itr && ch != ' '){
             found= TRUE;
             cur_opt = i;
            }
          }
        }
          if (ch==13){
           found = TRUE;
           expert = FALSE;
          }
```

```
            ch=' ';                    cur_x=2;cur_y=2;


        for(i=0;i< nu_main;i++){
            if(i == cur_opt) attribute= hi_attr;
             else attribute= nor_attr;
              print(cur_x,cur_y,m_menu[i].head);

          cur_x= cur_x+strlen(m_menu[i].head)+3;
         }

    if (!expert) found = TRUE;

    if (found){

      ext =(pull_down(m_menu,nu_sub,cur_opt)); /* pull-down options */
      if (ext == 27) expert = TRUE;
      if (ext == 'r' || ext == 'l') expert = FALSE;
      if (ext=='r') cur_opt = cur_opt+1;
      if (ext =='l') cur_opt = cur_opt -1;
      ch= ' ';
      ext= ' ';

     }

    if(!found){
     ch=' ';
     get_key(&ch,&ext);

     ch=toupper(ch);

   }
            if (ch==27) return;

            if (ext =='r' || ext == 'l') expert = 0;
            if (ext == 'r')  cur_opt = cur_opt +1;
            if (ext == 'l')  cur_opt = cur_opt -1;
            if (cur_opt >= nu_main) cur_opt =0;
            if (cur_opt < 0) cur_opt = nu_main-1;
            ext=' ';
            found=0;

   } /* end for(;;) */

}   /* end function */


int pull_down(m_menu,nu_sub,position)
struct menu_str m_menu[];
int position;

{
extern unsigned int page;extern unsigned int attribute;
char ch=' ',ltr;
int ext=' ',hi_attr,nor_attr;
int i,j,tx,ty,start,width,nu_opt,cur_opt=0, found= FALSE;

nu_opt = nu_sub;

/* nu_sub = number of possible pull-down options */
/* find out how many are in use */

for(i=0;i<nu_opt;i++){
  if (m_menu[position].body[i][0] == '\0'){
   nu_opt = i;
   break;
  }

}
```

continued

```
  if (mon_type==1){
    hi_attr=  set_color(BLACK,CYAN);
    nor_attr= set_color(WHITE,BLACK);
  }
    else{
     hi_attr = REVERSE;
     nor_attr = NORMAL;
    }

 attribute = nor_attr;

start=2;        /* figure where to draw pull-down box */
               /* 2 is column to start 1st box */
               /* add up length of menu heads */

  for(i=0; i< position; i++)  start= start+strlen(m_menu[i].head)+3;


  width=0;       /* figure max length of window */

  for (i=0;i< nu_opt;i++){
   if (strlen(m_menu[position].body[i]) > width){
    width= strlen(m_menu[position].body[i]);
   }
  }

                        /* move box to left if          */
                        /* it will spill off right side */

 if(start+width+1>80) start = 80-width-2;


  win_save ('s');

 if (mon_type ==1) attribute = set_color (YELLOW,BLACK);
 make_window(start++,3,width,nu_opt,0); /*make a window */
 attribute = nor_attr;

tx=start;ty=4;                          /* reposition for writing */

  for(;;){
         for(i=0;i< nu_opt;i++){
           if(i == cur_opt) attribute= hi_attr;
            else attribute= nor_attr;
           print(tx,ty++,m_menu[position].body[i]);
         };

         attribute = nor_attr;

          if(found ) {
             win_save('r');      /* remove box */

                /* if you want more than 5 menu options */
                /* change this next switch statement    */

             switch (cur_opt){       /* call function */
              case 0: (*m_menu[position].fun1)() ;break;
              case 1: (*m_menu[position].fun2)() ;break;
              case 2: (*m_menu[position].fun3)() ;break;
              case 3: (*m_menu[position].fun4)() ;break;
              case 4: (*m_menu[position].fun5)() ;break;

             }

          /* found = FALSE; */
          if (kbhit()) getch();  /* make sure keyboard buffer is clear */
           return(' ');
          }

        tx=start;ty=4;
       get_key(&ch,&ext); ch=toupper(ch);  /* get a character */
           if (ext == 'd')  cur_opt = cur_opt +1;
           if (ext == 'u')  cur_opt = cur_opt -1;
           if (cur_opt >= nu_opt) cur_opt =0;
           if (cur_opt < 0) cur_opt = nu_opt-1;
```

```
          if (ch== 13) found = TRUE;

        for(i=0;i<nu_opt;i++){   /* does it match an option? */
         j=0;
         while(ltr = m_menu[position].body[i][j++]){
            if ( ch==ltr){
             cur_opt = i;
             found = TRUE;
            }
         }
        }

           if (ext=='l'|| ext=='r') break;
           if (ch==27){          /* EXIT IF ESCAPE KEY */          ext = ch;
            break;
           }
           ext=' ';ch=' ';
  } /* end for(;;)*/
  win_save('r');
  return (ext);
}


void make_help()
{

extern unsigned int page,attribute;

 page=1;


  print(1,1,"HELLO - THIS IS A SAMPLE OF AN INSTANT HELP SCREEN.");
  print(10,5,"THIS SCREEN WAS PRINTED TO THE SECOND PAGE OF GRAPHICS");
  print(10,7,"WHILE YOU WERE LOOKING AT THE MAIN MENU.");
  print(10,9,"THIS HELP SCREEN CAN BE LEFT UNDISTURBED");
  print(10,11,"AND REDISPLAYED AT ANY TIME.");
  print (1,20,"PLEASE TOUCH ANY KEY TO RETURN TO THE MAIN MENU.");


 page=0;

}


void make_inst()
{
 extern unsigned int attribute;

  if (mon_type==1)
   attribute=  set_color(GREEN,BLACK);
    else
     attribute = NORMAL;

 print (1,4,"INSTRUCTIONS:");
 print (1,6,"EXPERT MODE: Select by touching the key which represents each
option.");
 print (15,7,"(the capital letter)");

 print (1,10, "ASSIST MODE: Pull-down menu by touching 'enter' or a cursor
key.");
 print (14,11,"Select by highlighting with cursor keys- then touch return");
 print (14,13,"Return to Expert mode by touching 'escape'");

 print (1,15,"EXIT:        Touch 'Escape' while in expert mode.");
}

void mono_help()
{

  attribute=NORMAL;
  win_save('s');
```

```
  clear_window(1,4,80,21);
  print(1,7,"THIS IS A DEMONSTRATION OF A HELP SCREEN");
  print(1,9,"THIS TEXT WAS WRITTEN BY MEANS OF DIRECT MEMORY ADDRESS");
  print(1,10,"THE ORIGINAL SCREEN HAS BEEN SAVED AND WILL BE RESTORED ");
  print(1,11,"WHEN YOU EXIT THIS 'HELP' SCREEN");
  print(1,14,"PLEASE TOUCH ANY KEY TO CONTINUE");

   getch();
  win_save('r');
}



int demo()
{
int hit;
win_save('s');
make_window(20,10,40,5,1);

print(21,11,"Put your favorite routine here ");
print (21,14,"touch any key to return to menu");
getch(hit);

win_save('r');
}


int help()
{
 if (mon_type==1){     /*IF COLOR CARD FLIP PAGE TO */
    page=1;            /*SHOW TEXT ELSE WRITE TO CURRENT*/
    d_page();getch();  /* SCREEN*/
    page=0;d_page();

 } else mono_help();

}


                /* SCREEN-FUNCTION LIBRAY */


/* DECLARE THE EXTERN VARIABLES        */
/* PAGE,ATTRIBUTE AND MON_TYPE         */
/* (MONITOR TYPE)                      */
/* IN YOUR MAIN PROGRAM                */



/*** GOTOXY ***/                       /* PUTS CURSOR AT X,Y POSITION */
void gotoxy(x,y)                       /* ON SELECTED PAGE            */
unsigned int x,y;                      /* 1,1 IS UPPER LEFT CORNER    */

{
   extern unsigned int page;

   struct { unsigned int ax,bx,cx,dx,si,di,ds,es; } regist;
   if (x<1 || x>80) return;
   if (y<1 || y>25) return;
   x--;y--;                  /* DOS starts co-ordinates at 0,0 */

   regist.ax = 0x0200;
   regist.dx = (y<<8) | x ;
   regist.bx = page<<8;  /*page number*/

   int86(0x10, &regist,&regist);
}

/*** WHEREXY ***/        /* RETURNS THE X,Y POSITION OF CURSOR */
int wherexy(x,y)
int *x,*y;
```

```
{
   extern unsigned int page;

   struct { unsigned int ax,bx,cx,dx,si,di,ds,es; } regist;


   regist.ax = 0x0300;

   regist.bx = page<<8;  /*page number*/

   int86(0x10, &regist,&regist);
   *x=( (regist.dx & 0x00ff)+1);
   *y= ( ( (regist.dx & 0xff00)>>8)+1);

}

/*** d_page ***/                  /* DISPLAYS THE PAGE INDICATED  */
void d_page()                     /* BY EXTER VAR PAGE            */
                                  /* USE ONLY WITH COLOR CARD     */

{
   extern unsigned int page;
   struct { unsigned int ax,bx,cx,dx,si,di,ds,es; } regist;
   regist.ax = (0x0500|page);
   int86(0x10, &regist,&regist);
}


/*** WIN_SAVE ***/                 /* SAVES OR RESTORES PRIMARY  */
void win_save(action)              /* DISPLAY SCREEN.            */
int action;                        /* (PAGE 0 FOR COLOR DISPLAY) */
                                   /* 's' = SAVE                 */
{                                  /* 'r' = RESTORE              */
 extern unsigned int page;         /*  SAVES CURSOR POSITION TOO */
 extern unsigned int mon_type;     /*  MAY MAKE SNOW ON CGA      */
 int position;
 static int ptr;

static struct {
        int x;
        int y;
        unsigned int buffer [4000];
      } window[2];


   if (mon_type==1) position=0xb800;  /* COLOR CARD */
    else position=0xb000;             /* MONOCHROME */

   if (action=='s') {                 /* SAVE */
    if (ptr>1){
     ptr=2;
     return(0);
    }
                        /* peek is a lattice function */
                        /* could use pointer in larger */
                        /* memory model                */
    peek(position,0x00,&window[ptr].buffer,4000);  /* SAVE SCREEN     */
    wherexy(&window[ptr].x,&window[ptr].y);        /* SAVE CURSOR LOC */
    ptr++;

   }

   if (action=='r') {                 /* RESTORE */
    if(ptr <1){
      ptr = 0;
      return(0);

    }
    ptr-- ;

    poke(position,0x00,&window[ptr].buffer,4000); /* RESTORE SCREEN */
    gotoxy(window[ptr].x, window[ptr].y);         /* RESTORE CURSOR */
```

continued

```
    }

}


/*** SET_COLOR ***/                          /* CALL WITH FORGROUND  */
                                              /* AND BACKGROUND COLOR.*/
int set_color(foreground, background,)        /* RETURNS ATTRIBUTE.   */
  int foreground,background;
{
  return(background<<4|foreground);

}



/*** CLEAR_WINDOW ***/                       /* CALL WITH X,Y OF UPPER LEFT   */
void clear_window(x,y,width,height )          /* CORNER OF WINDOW AREA.        */
unsigned int x,y,width,height;                /* CLEARS DOWN AND TO RIGHT      */
                                              /* FOR WIDTH AND HEIGHT.         */
{                                             /* CLEARED WITH ACTIVE ATTRIBUTE */
   extern unsigned int page;                  /* USE ON DISPLAYED PAGE ONLY!   */
   extern unsigned int attribute;
   struct { unsigned int ax,bx,cx,dx,si,di,ds,es; } regist;
   x--;y--;
   regist.ax = 0x0600;
   regist.cx = (y<<8) | x ;
   regist.dx =(y+height-1) <<8 |x + width-1 ;
   regist.bx =(attribute<<8);
   int86(0x10, &regist,&regist);
}



/*** box ***/

void box (x,y,width,height,type)     /* type 0 = pull-down box */
int x,y,width,height,type;           /* type 1 = regular box   */

{
  int i,j,ctr,u_right,u_left;
  char string[82];

  if(type==0){                    /* following sets corners */
   u_left  =   '\xc2';
   u_right =  '\xc2';
  }

  if (type == 1){
   u_left = '\xda';
   u_right = '\xbf';
  }

     string[0]= u_left;
     for(i=1;i<=width;i++) string[i]='\xc4';
     string[i++]=u_right; string[i]='\0';
     print(x,y++,&string[0]);

       for (i=0; i<height;i++){
        print(x,y,"\xb3");
        print(x+width+1,y++,"\xb3");
       }

          string[0]='\xc0';
          for(i=1;i<=width;i++) string[i]='\xc4';
          string[i++]='\xd9';string[i]='\0';
          print(x,y++,&string[0]);
}


int what_mon()           /* RETURNS A 1 IF COLOR CARD PRESENT */
{                        /* RETURNS A 0 IF MONOCHROME CARD    */

char mode;         /* CHAR DEFINES AN 8 BIT INTEGER */
```

```
    peek(0x0040,0x0049,&mode,1);

    if (mode==7) return(0);
    else return(1);
}


void cls()          /* SAME AS DOS CLS */
{
clear_window(1,1,80,25);
gotoxy(1,1);
}


void make_window(x,y,width,height,type)  /* DRAWS AND CLEARS A BOX */
unsigned int x,y,width,height,type;

{
  box(x++,y++,width,height,type);        /* DRAW BOX       */
  clear_window(x,y,width,height);        /* CLEAR INTERIOR */
}



int cursor(size)    /* SETS CURSOR SIZE */
int size;           /* 0= no cursor, 1 = normal, 2= big cursor */
{

 struct { int ax,bx,cx,dx,si,di,ds,es; } regist;

 regist.ax= 0x0100;

    if (mon_type == 1 ){                /* COLOR */
     if (size == 0) regist.cx = 0x0f0f;
     if (size == 1) regist.cx = 0x0607;
     if (size == 2) regist.cx = 0x0107;
    }

    if (mon_type == 0 ){                /* MONOCHROME */     if (size == 0) regist.cx= 0x0f0f;
     if (size == 1) regist.cx = 0x0c0d;
     if (size == 2) regist.cx = 0x010d;
    }

   int86(0x10,&regist,&regist);

}



/*** PRINT ***/

void print(x,y,str)          /* A SWITCHER- ROUTES TO FAST_WRITE */
 unsigned int x,y;           /* OR TO DOS_PRT DEPENDING ON ARGV  */
 char *str;                  /* PASSED TO PROGRAM AND SORED IN   */
{                            /* EXTERN CHAR WR_METH              */
 extern char wrt_meth;

if (wrt_meth == 'f')                        /* FAST (DIRECT POKING) */
  if (wrt_meth == 'f') fast_write(x,y,str);

  if (wrt_meth == 's') {     /* SLOW (DOS METHOD) */
   gotoxy(x,y);
   dos_prt(str);
  }
}

/*** DOS_PRT ***/           /*  ASKS DOS TO WRITE A STRING WITH ATTRIBUTE   */
                           /*  DEFINED. AN ALTERNATIVE TO FAST WRITE IN    */
void dos_prt(str)          /*  THAT IT IS "WELL BEHAVED"(GOES THROUGH DOS) */
                           /*  SPECIFY PAGE AND SET CURSOR POSITION BEFORE */
char *str;                 /*  CALLING   */
{
   extern unsigned int page,attribute;
   unsigned int x,y;
   int c;
```

continued

```
   struct { unsigned int ax,bx,cx,dx,si,di,ds,es; } regist;
   wherexy(&x,&y);

   while (*str) {                        /* WHILE NOT EOF LET DOS WRITE CHAR */
      if (x>80){
         x=1;
         y=y+1;
       }
      if (y>25) break;
      gotoxy(x++,y);
      regist.bx = (page<<8|attribute);

      regist.cx = 1;
      regist.ax = 0x0900| *str++;
      int86(0x10, &regist, &regist);

  }  gotoxy(x,y);              /* put cursor at end of string */
}

/***FAST_WRITE***/                 /* DIRECTLY POKES STRING AT X,Y POSITION */
void fast_write(x,y,string)      /* USES ATTRIBUTE AND PAGE.              */
int x,y;                         /* MAY CAUSE SNOW ON SOME GRAPHIC CARDS  */
char *string;

{
 extern unsigned int page;
 extern unsigned int attribute;
 extern unsigned int mon_type;   /* MONITOR TYPE */

 int position,offset,orig;

    if (page <=3 && page >=0) offset = 4000*page+96*page;
      orig = offset;
      offset=offset+((y-1)*160)+(2*(x-1));
      position =0xb800;

    if (mon_type ==0) position=0xb000;

   while(*string){
        poke(position,offset,string++,1);        /* POKE CHARACTER */
        poke(position,offset+1,&attribute,1);    /* POKE ATTRIBUTE */
        offset=offset+2;

   }
         offset = offset- orig;                    /* FIGURE WHERE I AM */
         x= ((offset% 160)/2)+1 ;y= offset/160+1 ; /* AND MOVE CURSOR   */
         gotoxy (x,y);

}

/*** get_key ***/              /* READ A CHAR              */
                               /* RETURN CHARACTER IN CH   */
int get_key(ch,ext)            /* IF IT IS A FUNCTION KEY */
char *ch;                      /* RETURN FOLLOWING IN EXT */
int  *ext;                     /* UP-ARROW    = 'U'        */
                               /* DOWN-ARROW  = 'D'        */
{                              /* RIGHT-ARROW = 'R'        */
  *ch=getch();                 /* LEFT-ARROW  = 'L'        */

    if(!*ch){
      *ext=getch();

       switch (*ext){
         case 'H' :*ext ='u';break; /*  up    */
         case 'P' :*ext ='d';break; /*  down    */
         case 'M' :*ext ='r';break; /*  right   */
         case 'K' :*ext ='l';break; /*  left    */
         case 'G' :*ext ='h';break; /*  home    */
         case 'O' :*ext ='e';break; /*  end     */
         case 'R' :*ext ='I';break; /*  insert  */
         case 'S' :*ext ='D';break; /*  delete  */       }
    }
}
```

LIST1.PS Contributed by: Denis G. Pelli
BINARY "Putting Postscript to Work," by Denis G. Pelli, May 1987, page 185.

```
%!
% Produce Figure 1
/preSloan {gsave currentpoint translate 0.2 0.2 scale
   newpath 0 0 moveto 0 5 lineto 5 5 lineto 5 0 lineto
closepath clip newpath
   1 setlinewidth 2 setlinecap 0 setlinejoin} def
/postSloan {stroke grestore 2 0 rmoveto} def
/R {preSloan 0.5 0.5 moveto 0 4 rlineto 3.5 3.5 1 90 270
arcn
  -3 0 rlineto 4.44370 0 moveto -0.97547 2 rlineto
postSloan} def
/V {preSloan 0 6.34629 moveto 2.5 -6.25 rlineto 2.5 6.25
rlineto postSloan} def
50 700 moveto
40 40 scale
R
V
showpage
```

LIST2.PS Contributed by: Denis G. Pelli
BINARY "Putting Postscript to Work," by Denis G. Pelli, May 1987, page 185.

```
%!
% Produce Figure 2

% path of face
/face {39  210 moveto 108  210 lineto 155  181 lineto 167
     146 lineto 155  117 lineto 155  89 lineto 145  55 lineto
     134  27 lineto 116  8 lineto 78  7 lineto 46 10 lineto 23
     30 lineto 7  71 lineto 4  110 lineto
     -3  136 lineto 4  174 lineto 39  210 lineto 1  130 moveto
     36  143 lineto 57  162 lineto 71  182 lineto 85  160 lineto
     112  141 lineto 158  126 lineto 24  121 moveto 45  127
     lineto 60  125 lineto 24  108 moveto 34  116 lineto 52  116
     lineto 61  109 lineto 52  101 lineto 35  101 lineto 24  108
     lineto 105  107 moveto 115  115 lineto 131  115 lineto 141
     107 lineto 134  99 lineto 117  98 lineto 105  107 lineto
     131  126 moveto 110  126 lineto 100  122 lineto 92  113
     lineto 87  97 lineto 89  83 lineto 92  67 lineto 61  44
     moveto 71  37 lineto 79  39 lineto 91  36 lineto 98  40
     lineto 107  37 lineto 112  43 lineto 75  52 moveto 84  56
     lineto 94  52 lineto 102  57 lineto} def

% Draw high-pass ribbon
/draw {  gsave 0 setgray stroke grestore
  gsave currentlinewidth 2 div setlinewidth 1 setgray stroke
     grestore       newpath} def

% Set up screen, etc.
120 currentscreen 3 -1 roll pop setscreen % best at 2540/inch
0.69 setgray clippath fill 0 setgray      % best at 2540/inch
%30 currentscreen 3 -1 roll pop setscreen % best at 300/inch
%0.58 setgray clippath fill 0 setgray     % best at 300/inch
1 setlinecap
10 setlinewidth
30 560 translate

% Ever smaller faces across page
9{
```

continued

```
  face
  draw
  190 0 translate
  .707 .707 scale
} repeat

showpage
```

LIST3.PS Contributed by: Denis G. Pelli
BINARY "Putting Postscript to Work," by Denis G. Pelli, May 1987, page 185.

```
%!
% Produce Figure 4
/inch {72 mul} def
/width 4.54 inch def
50 400 translate
width width scale % make square image of desired width
/printerresolution
  72 0 matrix defaultmatrix dtransform dup mul exch dup mul add sqrt
def
/screen printerresolution 16 div def
screen 127 gt {/screen 127 def} if
screen currentscreen 3 -1 roll pop setscreen
/n width screen mul 72 div 2 sqrt mul 0.99 add cvi def % number of cells
across image
/nx n 2 mul def
/ny n 2 idiv def
/fbase 0.5 n mul 1.0 nx div exp def
/fa 360  nx div fbase ln div def
/c 1 def
/cbase 0.003 1.0 ny div exp def
/S nx string def

nx ny 8 [nx 0 0 ny 0 0]
{/c c cbase mul def /f fa def
  0 1 nx 1 sub{S exch f sin c mul 1.0 add 126.5 mul cvi put /f f fbase mul
def}for
  S}image

showpage
```

LIST4.PS Contributed by: Denis G. Pelli
BINARY "Putting Postscript to Work," by Denis G. Pelli, May 1987, page 185.

```
%!
% Produce Figure 5
50 750 moveto
/width 250 def
/size width 6 div def
/cuberoot2 2 1 3 div exp def
width 2 div size -1.5 mul rmoveto

/showrow
  {gsave
  FontDirectory /Sloan known
    {/Sloan findfont size scalefont setfont}
    {/Helvetica findfont size 1.8 mul scalefont setfont}
  ifelse
  dup stringwidth pop -2 div 0 rmoveto
  show
```

```
   grestore
   /size size cuberoot2 div def
   0 size -2 mul rmoveto
 } def

 (Z C H) showrow
 (O S H) showrow
 (H K N) showrow
 (O Z S) showrow
 (D H C) showrow
 (C N O) showrow
 (K D V) showrow
 (S H O) showrow
 (D R H) showrow
 (K N Z) showrow
 (R H K) showrow
 (C H O) showrow
 (N K S) showrow
 (C Z R) showrow
 (S O K) showrow
 (D N C) showrow
 (C H D) showrow
 (R O H) showrow
 (S N D) showrow
 (O C H) showrow
 (Z H K) showrow
 (C V H) showrow
 (V N Z) showrow
 (S C K) showrow
 (N D C) showrow
 (C N D) showrow
 (O V N) showrow
 (Z H V) showrow
 (H S K) showrow
 (D N O) showrow
 (N S K) showrow
 (D C Z) showrow
 (V H K) showrow
 (S K V) showrow
 (V R K) showrow
 (V R S) showrow
 (K Z N) showrow

showpage
```

LIST5.PS Contributed by: Denis G. Pelli
BINARY "Putting Postscript to Work," by Denis G. Pelli, May 1987, page 185.

```
%!
% Produce Figure 6
50 750 moveto
/width 250 def
/size width 6 div def
width 2 div size -1.5 mul rmoveto
FontDirectory /Sloan known
  {/Sloan findfont size scalefont setfont}
  {/Helvetica findfont size 1.8 mul scalefont setfont}
ifelse
/printerresolution
  72 0 matrix defaultmatrix dtransform dup mul exch dup mul add sqrt
def
/screen printerresolution 16 div def
screen 127 gt {/screen 127 def} if
screen currentscreen 3 -1 roll pop setscreen
/c 1 def

/setcontrast {1 sub neg setgray} def
```

continued

```
/showrow
  {gsave
  dup stringwidth pop -2 div 0 rmoveto
  c setcontrast
  show
  grestore
  /c c 2 div def
  0 size -2 mul rmoveto
} def

(N C R) showrow
(C H V) showrow
(Z R H) showrow
(S H N) showrow
(V D K) showrow
(N K Z) showrow
(S Z O) showrow
(R D N) showrow
(R Z S) showrow
(Z R N) showrow
showpage
```

June

LISTING1.BAS Contributed by: Paul D. Bourke
Programming Project: "A Contouring Subroutine," by Paul D. Bourke. June, page 143.

```
REM     Input variables to CONREC
REM     d(0:iub,0:jub)     'Matrix for the data surface
REM     iub, jub     'Index bounds of the data array
REM     x(0:iub)     'Data array for column coordinates
REM     y(0:jub)     'Data array for row coordinates
REM     nc     'Number of contour levels
REM     z(0:nc-1)     'Contour levels in increasing order
REM     VECOUT     'An external subroutine to plot the contour lines
REM     False and true boolean values
REM
conrec:
REM     Local declarations for CONREC
DIM h(4)     'Relative heights of the box above contour
DIM ish(4)     'Sign of h()
DIM xh(4)     'x coordinates of box
DIM yh(4)     'y coordinates of box
DIM im(3)     'Mapping from vertex numbers to x offsets
im(0)=0 : im(1)=1 : im(2)=1 : im(3)=0
DIM jm(3)     'Mapping from vertex numbers to y offsets
jm(0)=0 : jm(1)=0 : jm(2)=1 : jm(3)=1
DIM castab(2,2,2)     'Case switch table
DATA 0, 0, 8, 0, 2, 5, 7, 6, 9, 0, 3, 4, 1, 3, 1, 4, 3, 0, 9, 6, 7, 5, 2, 0, 8, 0, 0
FOR k=0 TO 2 : FOR j=0 TO 2 : FOR i=0 TO 2
    READ castab(k,j,i)
NEXT i : NEXT j : NEXT k
REM
REM     Check the input parameters for validity
prmerr=false
IF (iub=<0 OR jub=<0) THEN prmerr=true
IF (nc<=0) THEN prmerr=true
FOR k=1 TO nc-1 : IF (z(k)<=z(k-1)) THEN prmerr=true : NEXT k
IF (prmerr) THEN msg$="Error in input parameters" : RETURN
REM
REM     Scan the array, top down, left to right
REM     ================================
FOR j=jub-1 TO 0 STEP -1
FOR i=0 TO iub-1
REM     Find the lowest vertex
IF (d(i,j)<d(i,j+1)) THEN dmin=d(i,j) ELSE dmin=d(i,j+1)
IF (d(i+1,j)<dmin) THEN dmin=d(i+1,j)
IF (d(i+1,j+1)<dmin) THEN dmin=d(i+1,j+1)
REM     Find the highest vertex
IF (d(i,j)>d(i,j+1)) THEN dmax=d(i,j) ELSE dmax=d(i,j+1)
IF (d(i+1,j)>dmax) THEN dmax=d(i+1,j)
IF (d(i+1,j+1)>dmax) THEN dmax=d(i+1,j+1)
IF (dmax<z(0) OR dmin>z(nc-1)) THEN GOTO noneinbox
REM     Draw each contour within this box
FOR k=0 TO nc-1
IF ((z(k)<dmin) OR (z(k)>dmax)) THEN GOTO noneintri
FOR m=4 TO 0 STEP -1
IF m>0 THEN h(m)=d(i+im(m-1),j+jm(m-1))-z(k) : xh(m)=x(i+im(m-1)) :
yh(m)=y(j+jm(m-1))
IF m=0 THEN h(0)=(h(1)+h(2)+h(3)+h(4))/4 : xh(0)=(x(i)+x(i+1))/2 :
yh(0)=(y(j)+y(j+1))/2
IF h(m)>0 THEN ish(m)=2 : ELSE IF (h(m)<0) THEN ish(m)=0 : ELSE ish(m)=1
NEXT m
REM     Scan each triangle in the box
FOR m=1 TO 4
m1=m : m2=0 : m3=m+1 : IF (m3=5) THEN m3=1
case=CINT(castab(ish(m1),ish(m2),ish(m3)))
IF (case=0) THEN GOTO case0
```

continued

```
ON case GOTO case1,case2,case3,case4,case5,case6,case7,case8,case9
REM    Line between vertices m1 and m2
case1: x1=xh(m1) : y1=yh(m1) : x2=xh(m2) : y2=yh(m2)
           GOTO drawit
REM   Line between vertices m2 and m3
case2: x1=xh(m2) : y1=yh(m2) : x2=xh(m3) :y2=yh(m3)
           GOTO drawit
REM    Line between vertices m3 and m1
case3: x1=xh(m3) : y1=yh(m3) : x2=xh(m1) : y2=yh(m1)
           GOTO drawit
REM    Line between vertex m1 and side m2-m3
case4: x1=xh(m1) : y1=yh(m1)
           x2=(h(m3)*xh(m2)-h(m2)*xh(m3))/(h(m3)-h(m2))
           y2=(h(m3)*yh(m2)-h(m2)*yh(m3))/(h(m3)-h(m2))
           GOTO drawit
REM    Line between vertex m2 and side m3-m1
case5: x1=xh(m2) : y1=yh(m2)
          x2=(h(m1)*xh(m3)-h(m3)*xh(m1))/(h(m1)-h(m3))
          y2=(h(m1)*yh(m3)-h(m3)*yh(m1))/(h(m1)-h(m3))
          GOTO drawit
REM    Line between vertex m3 and side m1-m2
case6: x1=xh(m3) : y1=yh(m3)
          x2=(h(m2)*xh(m1)-h(m1)*xh(m2))/(h(m2)-h(m1))
          y2=(h(m2)*yh(m1)-h(m1)*yh(m2))/(h(m2)-h(m1))
          GOTO drawit
REM    Line between sides m1-m2 and m2-m3
case7: x1=(h(m2)*xh(m1)-h(m1)*xh(m2))/(h(m2)-h(m1))
           y1=(h(m2)*yh(m1)-h(m1)*yh(m2))/(h(m2)-h(m1))
           x2=(h(m3)*xh(m2)-h(m2)*xh(m3))/(h(m3)-h(m2))
           y2=(h(m3)*yh(m2)-h(m2)*yh(m3))/(h(m3)-h(m2))
           GOTO drawit
REM    Line between sides m2-m3 and m3-m1
case8: x1=(h(m3)*xh(m2)-h(m2)*xh(m3))/(h(m3)-h(m2))
           y1=(h(m3)*yh(m2)-h(m2)*yh(m3))/(h(m3)-h(m2))
           x2=(h(m1)*xh(m3)-h(m3)*xh(m1))/(h(m1)-h(m3))
           y2=(h(m1)*yh(m3)-h(m3)*yh(m1))/(h(m1)-h(m3))
           GOTO drawit
REM    Line between sides m3-m1 and m1-m2
case9: x1=(h(m1)*xh(m3)-h(m3)*xh(m1))/(h(m1)-h(m3))
           y1=(h(m1)*yh(m3)-h(m3)*yh(m1))/(h(m1)-h(m3))
           x2=(h(m2)*xh(m1)-h(m1)*xh(m2))/(h(m2)-h(m1))
           y2=(h(m2)*yh(m1)-h(m1)*yh(m2))/(h(m2)-h(m1))
drawit:
           CALL vecout(x1,y1,x2,y2,z(k))
case0: NEXT m
noneintri: NEXT k
noneinbox: NEXT i : NEXT j
RETURN
```

LISTING2.BAS Contributed by: Paul D. Bourke
Programming Project: "A Contouring Subroutine," by Paul D. Bourke. June, page 143.

```
CLS: PRINT "CONREC example. Contour the function"
PRINT: PRINT "f(x,y)=sin((x^2+y^2)^.5) + ((x-c)^2+y^2)^-.5"
PRINT: PRINT "letting x and y range from -2 pi to +2 pi."
PRINT "Building the data set now. Please wait."
OPTION BASE 0      'Lower bound of zero for all array indices
pi=3.141592654#
true=-1 : false=0
ilength=319 : jlength=199  'Dimensions of the output contour plot axes (full
screen in CGA mode)
imin=0  :jmin=199     'Coordinates of the left bottom corner
iub=30 : jub=30 : nc=10  'Number of grid intervals and contour levels
DIM d(iub,jub),x(iub),y(jub)     'Data array
DIM z(nc-1)     'Contour array
```

```
REM    Define the function and the coordinates
FOR i=0 TO iub  'Check at all x-grid levels
    ix=2*pi*(2*i-iub)/iub  'ix ranges from -2 pi to + 2 pi
    FOR j=0 TO jub  'Check at all y-grid levels
        jy=2*pi*(2*j-jub)/jub  'jy ranges from -2 pi to + 2 pi
        r=SQR(ix^2+jy^2)
        d(i,j)=SIN(r)+.5/SQR((ix+3.05)^2+iy^2)
    NEXT j
    x(i)=i*ilength/iub+imin 'Scale x(i) to span plot area
NEXT i
FOR j=0 TO jub
    y(j)=jmin-j*jlength/jub 'Scale y(i) to span plot area
NEXT j
FOR i=0 TO nc-1
    z(i)=(i-5)/5   'Contour levels at -1,-.8,...,1
NEXT i

CLS: SCREEN 1,0      'CGA screen 320 x 200
LINE (imin,jmin-jlength)-(imin+ilength,jmin),,b  'Use a box for axes
GOSUB conrec
IF NOT(prmerr) THEN PRINT : PRINT : PRINT msg$;
WHILE LEN(INKEY$)=0 : WEND    'Any key to stop
END
```

LISTING3.BAS Contributed by: Paul D. Bourke
Programming Project: "A Contouring Subroutine," by Paul D. Bourke. June, page 143.

```
CLS: PRINT "CONREC example. Graph the equipotential lines"
print "around two charg particles by contouring the function"
print: print "V(x,y)=q1/r1 - q2/r2"
print "letting x and y range from -4 to 4."
OPTION BASE 0 'Lower bound of zero for all array indices
pi=3.141592654#
true=-1 : false=0
ilength=319 : jlength=199    'Dimensions of the output contour plot axes (full
screen in CGA mode)
imin=0  :jmin=199    'Coordinates of the left bottom corner
iub=30 : jub=30 : nc=8 'Number of grid intervals and contour levels
DIM d(iub,jub),x(iub),y(jub)    'Data array
DIM z(nc-1)    'Contour array
REM
REM    Define the function and the coordinates
a=1.5 : q1=1 : q2=-4  'Charge q1 is at -a; q2 is at +a
FOR i=0 TO iub
    ix=4*(2*i-iub)/iub   'Range from -4 to 4
    FOR j=0 TO jub
        jy=4*(2*j-jub)/jub  'Range from -4 to 4
        r1=SQR((ix-a)^2+jy^2)
        r2=SQR((ix+a)^2+jy^2)
        d(i,j)=(q1/r1-q2/r2)
    NEXT j
    x(i)=i*ilength/iub+imin  'Scale x(i) to span plot area
NEXT i
FOR j=0 TO jub
    y(j)=jmin-j*jlength/jub  'Scale y(i) to span plot area
NEXT j
FOR i=0 TO nc-1 : z(i)=(i+1)/2 : NEXT i
REM
CLS: SCREEN 1,0    'CGA screen 320 x 200
LINE(imin,jmin-jlength)-(imin+ilength,jmin),,b 'Use a box for axes
GOSUB conrec
IF NOT(prmerr) THEN PRINT : PRINT : PRINT msg$;
WHILE LEN(INKEY$)=0 : WEND    'Any key to stop
CLS : WINDOW CLOSE 1
END
```

continued

LISTING4.BAS Contributed by: Paul D. Bourke
Programming Project: "A Contouring Subroutine," by Paul D. Bourke. June, page 143.

```
REM    User defined subroutine to plot vectors on whatever plotting
REM    device is availiable
REM
SUB vecout(xstart,ystart,xstop,ystop,clevel) STATIC
    LINE (xstart,ystart)-(xstop,ystop)
END SUB

** Total includes 76112K for BRUN3087
```

CLOCKSET.ASM Contributed by: Tim G. Hunkler
"68000 Machines: Atari 520ST Projects," Tim G. Hunkler. June, page 161.

```
******************
*                *
*  CLOCKSET.ASM  *   Set Time of Clock Cartridge
*                *   <=> PUBLIC DOMAIN 09-SEP-86 <=>
******************
*
*** created:  03-AUG-86 by  Tim Hunkler / Solar Powered Software
*
ROM3    equ     $FA0000         ; ROM3 select strobe
ROM4    equ     $FB0000         ; ROM4 select strobe
READ    equ     0               ; control for READ access (A6=0)
WRT     equ     64              ; control for WRITE access (A6=1)
*
* --- Function call definitions for GEM/AES calls
*
APPL_INI        equ 0           ; Application initialization
APPL_EXI        equ 1           ; Application exit
OBJC_DRA        equ 2           ; Object draw
FORM_DO         equ 3           ; Form:  Do
FORM_DIA        equ 4           ; Form:  Dialog
FORM_ALE        equ 5           ; Form:  Alert
FORM_CEN        equ 6           ; Form:  Center
GRAF_MOU        equ 7           ; Graphics:  Mouse Form
*
* --- Offset definitions for dynamic storage
*
params  equ     $0000   ; table of array pointers
control equ     $0018   ; array of control counts
global  equ     $0022   ; array of global variables
int_in  equ     $0040   ; array of integers in
int_out equ     $0060   ; array of integers out
addr_in equ     $006E   ; array of addresses in
addr_out equ    $0072   ; array of addresses out
buffer  equ     $0076   ; buffer to receive clock chip data
ap_id   equ     $0026   ; application ID, one of the globals
stack   equ     $0486   ; stack area (grows downward)
xx      equ     $048A   ; temp area to receive x,y,w,h info
sz_stor equ     $0492   ; size of dynamic storage needs

* ============================
* ----- PROGRAM ENTRY POINT -----
* ============================

x:      lea     storage(pc),a5  ; load base of dynamic storage
        lea     stack(a5),sp    ; establish a stack
*
* --- release all memory not needed
```

```
*
        move.l  #(storage-x+256+sz_stor),-(sp) ; push num bytes to keep
        pea     x-256(pc)       ; push address of memory to keep
        clr.w   -(sp)           ; push filler
        move.w  #74,-(sp)       ; push function code = memory shrink
        trap    #1              ; call the system
        lea     12(sp),sp       ; fix the stack
*
* --- initialize GEM/AES parameter block
*
        lea     params+24(a5),a6        ; address end of block
        lea     control(a5),a0          ; first pntr
        lea     global(a5),a1           ; second pntr
        lea     int_in(a5),a2           ; third pntr
        lea     int_out(a5),a3          ; fourth pntr
        lea     addr_in(a5),a4          ; fifth pntr
        lea     addr_out(a5),a5         ; sixth pntr
        movem.l a0-a5,-(a6)             ; fill data block
*
* --- zero some areas in =global=
*
        moveq   #14,d0          ; loop for 15 words
zgbl:   clr.w   (a1)+           ; clear entire array
        dbf     d0,zgbl
*
* --- initialize base and data pointers and start program
*
        lea     x(pc),a6        ; A6 will be program base register
        lea     storage(pc),a5  ; A5 will be data storage base register
*
* --- begin regular program code
*
START:  bsr     _appl_in        ; initialize application
        bsr     _mouse          ; change mouse pointer to an arrow
*
* --- read info from the clock cartridge
*
rdcart: lea     buffer(a5),a2   ; load addr of buffer to be filled
        bsr     r_clock         ; read time from the clock cartridge
        beq.s   hwok            ; branch if all went ok
*
* --- clock's time was bad or the hardware is not working
*
        lea     warning(pc),a0  ; address warning message
        bsr     _alert          ; and display the message
*
* --- process dialog menu
*
hwok:   bsr     format          ; format time/date into dialog strings
        bsr     dialog          ; display and process the dialog
*
* --- determine which exit button was used
*
        moveq   #1,d3           ; outer box = object #1
        bsr     tstnclr         ; was outer box selected?
        bne.s   rdcart          ; yes, then update the time displayed
        moveq   #12,d3          ; SET = object #12
        bsr     tstnclr         ; was SET button selected?
        beq.s   if13            ; no, branch
        bsr     set_clk         ; yes, change time
        bra.s   rdcart

if13:   bsr     tstnclr         ; was button = EXIT?
        beq.s   rdcart          ; no, then update menu
*
* --- exit this program
*
        bsr     _appl_ex        ; notify GEM of application exit

        clr.w   -(sp)           ; function code = exit
        trap    #1              ; call TOS and never return!

* ====================================
* ===== w_clock :  write to clock =====
* ====================================
```

continued

```
*
*       in:  D0.B = data to be written
*            D1.W = register address x 2 to be written to
*
*      out:  A3.L = pointer to clock address latch
*            A4.L = pointer to clock chip select
*            D1.W = increased by 2
*            D0.W = data present prior to write
*            cc's : set by D0
*
*  changed:  A0,D2
*
*  This subroutine is used to write data to the clock chip.  The data
*  to be written is passed in the lower 4 bits of D0 and 2 times the
*  clock register to be accessed in passed in D1.
*
w_clock:
        lea     ROM3,a3         ; LATCH address
        lea     ROM4,a4         ; CS low strobe
*
* --- form the clock register address
*
wcep:   lea     0(a3,d1.w),a0   ; form proper address
*
* --- mask off 4 bit data and align it into bits 12..15
*
        moveq   #15,d2          ; load mask
        and.w   d0,d2           ; mask of 4 bit data item
        ror.w   #4,d2           ; rotate to upper bits
*
* --- latch address, data, and enable writes to the chip
*
        move.b  WRT(a0,d2.1),d0 ; read old data, latch new data
*
* --- turn chip select on to start the write operation
*       tst.w   (a4)            ; set chip select low
        addq.w  #2,d1           ; update register address
*
* --- turn off write enable, turn off chip select, but hold data
*
        tst.b   READ(a0,d2.1)   ; set chip select high
        andi.w  #$000F,d0       ; mask off 4 bit data and set cond codes
        rts

* ====================================
* ===== r_clock : read from clock =====
* ====================================
*
*       in:  A2.L = address of 16 byte buffer to be filled
*
*      out:  D0.L = 0 if all OK, else -1
*            cc's : set by D0
*            (A2) ---> [ flag bits    ]  leap count, AM/PM, 12/24 hr.
*                 +1   [ day of week  ]  1=Sunday
*                 +2   [ year x 10    ]  00 = 1980, 01 = 1981, etc.
*                 +3   [ year x 1     ]
*                 +4   [ month x 10   ]
*                 +5   [ month x 1    ]
*                      [ day x 10     ]
*                      [ day x 1      ]
*                      [ hour x 10    ]
*                      [ hour x 1     ]
*                      [ minute x 10  ]
*                      [ minute x 1   ]
*                      [ second x 10  ]
*                      [ second x 1   ]
*                 +14  [ .1 seconds   ]
*                 +15  [ junk byte    ]
*
*  changed:  A0,A4,D1,D2
*
*  This subroutine reads all time and date registers from the clock chip
*  and fills a 16 byte buffer whose address was passed in A2.  As the
*  information is read it is added up.  Invalid sums typically indicate
*  a dead battery or a clock cartridge which is not present and are
*  indicated by returning a -1 in register D0 and the condition codes.
```

```
*  A data changed bit is checked and if the time changed while the read
*  operation was in progress the operation is repeated.
*
r_clock:
        lea     ROM4,a4         ; set pointer for ROM4 strobe
        lea     ROM3,a0         ; set pntr to register 0
        lea     16(a2),a2       ; point to end of buffer to fill
        moveq   #0,d1           ; set accumulator to zero
*
* --- preload the register address in the latch
*
        tst.w   (a0)+           ; set register selection to 0
*
* --- loop and read clock registers 0..15*
        moveq   #15,d2          ; loop count = 16
fetch:  tst.w   (a4)            ; set chip select on
        moveq   #15,d0          ; load 4 bit data mask
        and.w   (a0)+,d0        ; get data, set next addr, select off
        move.b  d0,-(a2)        ; store data in buffer
        add.w   d0,d1           ; add up all data values
        dbf     d2,fetch        ; and repeat for 16 items
*
* --- retrieve data changed flag (zero indicates no change occurred)
*
        tst.w   (a4)            ; set CS low with register select = 0
        moveq   #8,d0           ; load mask
        and.w   (a0),d0         ; get flag, nxt addr = 2, select = off
*
* --- if no clock present we generally accumulate 16 x 15 = 240
*
        cmpi.w  #125,d1         ; does accumulation indicate bad data?
        ble.s   nwflg           ; no, branch
        moveq   #-1,d0          ; yes, set error flag
*
* --- test data changed flag and reread the time if necessary
*
nwflg:  tst.w   d0              ; was data changed (or an error)?
        bgt.s   r_clock         ; yes, then repeat the time reading
        rts                     ; no, exit with cond codes set

* ================================================
* ===== format:  format clock data buffer into dialog =====
* ================================================
*
*       in:  A2.L = addr of 16 byte buffer returned by "r_clock"
*      out:  none
*  changed:  A0,A1,A2,A3,D0,D1,D2,D3
*
*  This subroutine takes the contents of the 16 byte buffer filled
*  in by "r_clock" and uses it to format information in the dialog
*  menu.  The 7 radio buttons which indicate the day of the week are
*  set based on the day of the week value of 1..7.  The date is
*  formatted and the date string is formed.  And also the time is
*  formatted and the time string is formed.
*
format: lea     tree(pc),a3     ; address dialog tree
*
* --- clear radio buttons
*
        moveq   #6,d2           ; loop for 7 days of the week
        moveq   #5,d3           ; start with button for SUN
fmtclr: bsr     tstnclr         ; clear buttons for SUN..SAT
        dbra    d2,fmtclr       ; repeat till done
*
* --- set radio button for day of the week
*
        moveq   #7,d3           ; load 3 bit mask
        and.w   (a2)+,d3        ; get day of week, 1..7
        addq.w  #4,d3           ; convert to object number 5..11
        bsr     setsel          ; set button for SUN..SAT
*
* --- form date string:  MM/DD/YY
*
        lea     s_date+4(pc),a1 ; access date string
        move.b  (a2)+,(a1)      ; copy tens digit of year
```

continued

```
        addi.b   #8,(a1)+        ; modify for 1980 baseline
        move.b   (a2)+,(a1)+     ; copy ones digit of year

        subq.l   #6,a1           ; back up to access month
        move.b   (a2)+,(a1)+     ; move two digits of month
        move.b   (a2)+,(a1)+

        move.b   (a2)+,(a1)+     ; move two digits of day-of-month
        move.b   (a2)+,(a1)+
*
* --- form time string:  HH:MM:SS AM
*
        moveq    #'A',d2         ; preload AM/PM indicator as AM
        moveq    #15,d0          ; load a mask
        and.b    (a2)+,d0        ; get tens digit of hour
        mulu     #10,d0
        add.b    (a2)+,d0        ; add in ones digit, hours in D0.W

        lea      s_time+2(pc),a1 ; address minute field
        move.b   (a2)+,(a1)+     ; move two digits of minutes
        move.b   (a2)+,(a1)+

        move.b   (a2)+,(a1)+     ; move two digits of seconds
        move.b   (a2)+,(a1)+

        tst.w    d0              ; is it midnight to 1:00 am ?
        bne.s    ampm            ; no, skip ahead
        moveq    #12,d0          ; yes, change from 00:xx to 12:xx
        bra.s    AM

ampm:   cmpi.w   #12,d0          ; does hour indicate PM?
        blt.s    AM              ; for 00:00 to 11:59 no change
        beq.s    noon            ; for 12:00 to 12:59 change flag to 'PM'
        subi.w   #12,d0          ; for 13:00 to 23:59 reduce by 12 hours
noon:   moveq    #'P',d2         ; change flag to PM

AM:     move.b   d2,(a1)+        ; store A or P for AM/PM

        subq.l   #7,a1           ; point to hours field
        divu     #10,d0          ; split hours into tens and ones digit
        move.b   d0,(a1)+        ; store tens digit
        swap     d0              ; access ones digit
        move.b   d0,(a1)         ; store ones digit
        subq.l   #1,a1           ; adjust pntr to time string
        bsr.s    convrt          ; convert binary into characters

        lea      s_date(pc),a1   ; adjust pntr to date string
convrt: moveq    #5,d1           ; do 6 digitscvt:
        ori.b    #'0',(a1)+      ; convert byte into ASCII
        dbra     d1,cvt
        rts

* ======================================================
* ===== set_clk :  Set time/date in clock cartridge =====
* ======================================================
*
*       in:  A3.L = address of dialog tree
*      out:  none
*  changed:
*
*  This subroutine uses the time and date present on the dialog
*  menu to set the time and date for GEM and TOS.  It then converts
*  this information into the appropriate fields of the 16 byte table
*  filled in by 'r_clock' and uses this table to alter the time and
*  date of the MM58274 clock chip.
*
set_clk:
        bsr      set_time        ; set time for TOS and GEM
        bne.s    tsok            ; branch if all went ok

        lea      badtime(pc),a0  ; address error message
        bra      _alert          ; display alert then return

tsok:   lea      buffer-1(a5),a2 ;       ; access byte prior to buffer
        move.b   #1,(a2)+        ; install command = %0001
        clr.b    (a2)+           ; install command = %0000
```

```
*
* --- figure out which day of the week is indicated
*
        moveq   #5,d3           ; select button = SUN
sfdow:  bsr     tstnclr         ; is it selected?
        beq.s   sfdow           ; no, then try next button
        subq.w  #5,d3           ; convert SUN..SAT into 1..7
        move.b  d3,(a2)+        ; and store
*
* --- convert the date string into binary nibbles
*
        lea     s_date+4(pc),a0 ; access tens digit of year
        subq.b  #8,(a0)         ; convert 1980 into 00
        move.b  (a0)+,(a2)+     ; move two year digits
        move.b  (a0)+,(a2)+

        subq.l  #6,a0           ; access month
        move.b  (a0)+,(a2)+     ; move two month digits
        move.b  (a0)+,(a2)+

        move.b  (a0)+,(a2)+     ; move two day digits
        move.b  (a0)+,(a2)+
*
* --- convert the time string into binary nibbles
*
        lea     s_time(pc),a0   ; access time string
        bsr     conv2dig        ; convert hours into binary
        cmpi.b  #'A',4(a0)      ; is AM or PM indicated?
        beq.s   nnta            ; branch if AM
        cmpi.w  #12,d0          ; is hour between 12:00 and 12:59?
        beq.s   nnta            ; yes, then skip
        addi.w  #12,d0          ; adjust 1:00 into 13:00

nnta:   divu    #10,d0          ; split hours into digits
        move.b  d0,(a2)+        ; store tens digit of hours
        swap    d0              ; access ones digit
        move.b  d0,(a2)+        ; store ones digit of hours

        move.b  (a0)+,(a2)+     ; move two digits of minutes
        move.b  (a0)+,(a2)+

        move.b  (a0)+,(a2)+     ; move two digits of seconds
        move.b  (a0)+,(a2)+
*
* --- translate ASCII digits into binary
*
        moveq   #13,d0          ; loop 14 bytes
xclr:   andi.b  #15,-(a2)       ; convert ASCII digits to binary
        dbra    d0,xclr
*
* --- begin clock update
*
        lea     buffer+14(a5),a2        ; address end of time data
        moveq   #5,d0           ; command = stop clock
        moveq   #0,d1           ; address = 0

        bsr     w_clock         ; write command to stop clock

        moveq   #1,d0           ; command = set 24 hour mode
        moveq   #30,d1          ; addr = 15
        bsr     w_clock         ; write command

        moveq   #7,d0           ; command = interrupts off
        moveq   #0,d1           ; addr = 0
        bsr     w_clock         ; write command
*
* --- reprogram all registers from the table and start clock
*
        moveq   #14,d3          ; loop for 15 writes
        moveq   #4,d1           ; begin addr = 2 (seconds register)
sb2c:   move.b  -(a2),d0        ; get next time digit
        bsr     w_clock         ; change the time
        dbra    d3,sb2c
        rts
```

continued

```
* ========================================================
* ===== conv2dig :  convert decimal string into binary =====
* ========================================================
*
*       in:  A0.L = address of string
*      out:  D0.L = binary value, range = 0..99
*            A0.L = incremented by 2*            cc's : set by D0
*  changed:  D1
*
*  This subroutine converts the two digit ASCII string pointed to by
*  A0 into a binary value of the range 0..99.  The alternate entry
*  point 'tenx' forms D0.L = 10*D0 + D1
*
conv2dig:
        moveq   #15,d0          ; load mask for ten's digit
        moveq   #15,d1          ; load mask for one's digit
        and.b   (a0)+,d0        ; retrieve ten's digit
        and.b   (a0)+,d1        ; retrieve one's digit
*
* --- form the result:   10*D0 + D1
*
tenx:   add.l   d0,d0           ; 2x
        add.l   d0,d1           ; form 2x + y
        add.l   d0,d0           ; 4x
        add.l   d0,d0           ; 8x
        add.l   d1,d0           ; result = 8x + 2x + y = 10x + y
        rts

* ======================================
* ===== chk_time : check time string =====
* ======================================
*
*       in:  none
*      out:  CC's : Z=1 if invalid string
*  changed:  a0,a1,d0,d1,d2
*
*  This subroutine checks the time and date strings of the dialog for
*  valid digits.  If any invalid digit is present the condition codes
*  are set accordingly before exit.
*
t_vld:  dc.w    $0003   ; month x 10    = [0..1]
        dc.w    $03ff   ; month x 1     = [0..9]
        dc.w    $000f   ; date x 10     = [0..3]
        dc.w    $03ff   ; date x 1      = [0..9]
        dc.w    $0300   ; year x 10     = [8..9]
        dc.w    $03ff   ; year x 1      = [0..9]
        dc.w    -1
        dc.w    $0003   ; hour x 10     = [0..1]
        dc.w    $03ff   ; hour x 1      = [0..9]
        dc.w    $003f   ; minute x 10   = [0..5]
        dc.w    $03ff   ; minute x 1    = [0..9]
        dc.w    $003f   ; second x 10   = [0..5]
        dc.w    $03ff   ; second x 1    = [0..9]
*
chk_time:
        lea     s_date(pc),a0   ; address the date string
        lea     t_vld(pc),a1    ; point to table
        moveq   #11,d2          ; loop count = 6 + 6 digits
ct:     moveq   #15,d0          ; load mask
        and.b   (a0)+,d0        ; get lower nibble of digit
        move.w  (a1)+,d1        ; get validation bits
        bpl.s   cx              ; branch unless validation = -1
        lea     s_time(pc),a0   ; if -1, switch to time string
        bra.s   ct              ; and continue
cx:     btst    d0,d1           ; is digit valid?
        dbeq    d2,ct           ; loop if valid, stop if not
        rts                     ; return with cc's set

* ==================================
* ===== set_time : set the time =====
* ==================================
*
*       in:  none
*      out:  none
*  changed:
```

```
*
*  This routine checks the time and date strings of the dialog menu for
*  validity.  If the strings are valid the information in them is used
*  to generate the bit formats necessary to tell GEM and TOS what the
*  time and date are.  Then the time and date are changed via system
*  calls.
*
set_time:
        bsr.s   chk_time        ; is the time and date valid?
        beq.s   stxit           ; no, then exit
*
* --- convert date into BIOS format
*
        lea      s_date(pc),a0  ; address the date string
        bsr.s    conv2dig       ; pull off the month
        moveq    #$0F,d7        ; load 4 bit mask
        and.w    d0,d7          ; 0000 0000 0000 mmmm

        bsr.s   conv2dig        ; pull off the day of the month
        andi.w  #$001F,d0       ; mask down to 5 bits
        lsl.w   #5,d7           ; shift month bits left
        or.w    d0,d7           ; 0000 000m mmmd dddd

        bsr     conv2dig        ; pull off the year
        subi.w  #80,d0          ; convert 80 into 00
        andi.w  #$007F,d0       ; mask down to 7 bits
        ror.w   #7,d0           ; roll year to high end of word
        or.w    d0,d7           ; yyyy yyym mmmd dddd

        swap    d7              ; save 'date' in high order bits of D7
*
* --- convert time into BIOS format
*
        lea      s_time(pc),a0  ; address time string
        bsr     conv2dig        ; get HH
        move.w  d0,d7           ; 0000 0000 000h hhhh

        bsr     conv2dig        ; get MM

        cmpi.b  #'A',(a0)       ; are we AM or PM?        beq.s   useAM           ; branch if AM
        cmpi.w  #12,d7          ; are we 12:xx PM?
        beq.s   useAM           ; yes, then don't correct
        addi.w  #12,d7          ; else convert 1..11 into 13..23

useAM:  lsl.w   #6,d7           ; shift bits to make room for minutes
        andi.w  #$003F,d0       ; mask minutes down to 6 bits
        or.w    d0,d7           ; 0000 0hhh hhmm mmmm
*
* --- align time and set seconds to zero
*
        lsl.w   #5,d7           ; hhhh hmmm mmm0 0000
*
* --- tell GEM what the date and time is
*
        move.l  d7,-(sp)        ; pass [date:time] on stack
        move.w  #22,-(sp)       ; function code
        trap    #14             ; tell GEM what date/time is
        addq.l  #6,sp           ; fix stack
*
* --- tell TOS what the time is
*
        move.w  d7,-(sp)        ; push "time"
        move.w  #45,-(sp)       ; function code
        trap    #1              ; tell TOS what time is
        addq.l  #4,sp           ; fix stack
*
* --- tell TOS what the date is
*
        swap    d7              ; reorder [date:time] to [time:date]
        move.w  d7,-(sp)        ; push "date"
        move.w  #43,-(sp)       ; function code
        trap    #1              ; tell TOS what date is
        addq.l  #4,sp           ; fix stack
```

continued

```
        moveq    #-1,d7          ; set condition codes to indicate OK
stxit:  rts

* ================================================
* ===== dialog : display dialog and await exit =====
* ================================================
*
*       in:  A3.L = address of tree
*      out:  D3.W = object ID causing exit
*  changed:  A0, D0, ?
*
*  This subroutine reserves memory, draws the tree addressed by A3,
*  processes the dialog, erases the dialog, sets the button which caused
*  exit to non-selected, and frees up the reserved memory.  The object
*  number of the button which caused exit is returned in D3.
*
dialog: bsr      rsrv_win        ; reserve window space
        bsr      treedraw        ; draw the tree
        bsr      wfbutton        ; wait for response
        bra      free_win        ; erase window
* ================================================
* ===== tstnclr :  Test and Clear SELECTED bit =====
* ================================================
*
*       in:  A3.L = tree address
*            D3.W = object ID
*      out:  D3.W = incremented by 1
*            D0.W = object's state word anded with 1
*            cc's : set by D0.W
*  changed:  A0
*
*  This subroutine tests the SELECTED bit of the object number passed in
*  D3 and increments D3.  The state of the bit is used to set a
*  condition code before exit.
*
tstnclr:
        bsr.s    objba           ; get address
        moveq    #SELECTED,d0    ; load mask
        and.w    (a0),d0         ; get bit
        eor.w    d0,(a0)         ; clear bit if set
        addq.w   #1,d3           ; update
        tst.w    d0              ; set condition codes
        rts

* ============================================
* ===== objba    :  object bit address ========
* ============================================
*
*    input:  A3.L = address of tree
*            D3.W = object ID
*      out:  A0.L = address of object's state word
*  changed:  D0
*
*  This subroutine is used to form the address of the resource object's
*  STATE word when given the resource tree address and object number.
*
objba:   move.w  d3,d0           ; copy object number
         mulu    #24,d0          ; form 24 byte offset per object
         lea     10(a3,d0.1),a0 ; STATE word is 10 bytes into object
         rts

* ============================================
* ===== objbset :  object bit set ===========
* ============================================
*
*       in:  A3.L = address of tree
*            D3.W = object id
*            D4.W = bit mask
*      out:  A0.L = address of object's state word
*  changed:  none
*
*  This subroutine is used to 'OR' a bit mask with the STATE word of an
*  object in a resource tree.  The alternate entry point 'setsel' is
*  used when the 'SELECTED' bit is to be set.*
```

```
setsel:  moveq   #1,d4          ; load SELECTED bit mask
objbset: bsr.s   objba          ; form address
         or.w    d4,(a0)        ; set bits by ORing
         rts

* ========================================
* ===== rsrv_win : reserve window ==========
* ========================================
*
*       in:  A3.L = address of tree about to be displayed
*      out:  none
*  changed:  D0,D1,A0,A1
*
*  This routine first calls the FORM_CENTER routine to set the x,y,w,h
*  needed to center the window.  Then control drops into FREE_WIN which
*  reserves the memory needed for the window size
*
rsrv_win:
         lea     xx(a5),a1      ; addr temp area to receive x,y,w,h
         bsr     _form_ce       ; calculate window center
         moveq   #0,d0          ; pass code for reserving space
         bra.s   fwep           ; enter free_win routine

* ======================================================
* ===== free_win :  free up memory reserved earlier =====
* ======================================================
*
*       in:  none
*      out:  none
*  changed:  D0,D1,A0,A1
*
*  This subroutine frees up memory which was reserved earlier.  The
*  x,y,w,h clip rectangle was earlier stored at 'xx'.
*
free_win:
         moveq   #3,d0          ; use code = 3,
fwep:    lea     xx(a5),a0      ; load pointer to x,y,w,h area
         bra.s   _form_di       ; free memory

* ====================================
* ===== objdraw :  draw an object =====
* ====================================
*
*       in:  A3.L = address of tree
*            D3.W = object to start with
*      out:  none
*  changed:  ?
*
*  This subroutine draws the resource tree starting at the object number
*  given in D3 and for 5 levels of offspring.  The alternate entry point
*  'treedraw' starts at object number 0 for drawing an entire tree.
*
treedraw:
         moveq   #0,d3          ; for trees set object number to zeroobjdraw:
         lea     xx(a5),a0      ; address x,y,w,h
         moveq   #5,d1          ; pass 5 levels
         move.w  d3,d0          ; pass object number in d0
         bra     _objc_dr       ; draw

* ============================================
* ===== wfbutton :  wait for a button ==========
* ============================================
*
*       in:  A3.L = tree address
*            D0.W = object ID of editable text, 0 if none
*      out:  D3.W = ID of button causing exit
*  changed:  ?
*
*  This routine calls the FORM_DO routine which processes the user
*  interaction with a dialog.  Upon exit the object number which was
*  used to exit the dialog is returned in D3.
wfbutton:
         moveq   #0,d0          ; set object number to 0
wftext:  bsr.s   _form_do       ; process dialog
         move.w  d0,d3          ; assign to exitobj
         rts
```

continued

```
* ================================
* ===== _APPL_IN : initialize =====
* ================================
*
*       in:  none
*      out:  D0.W = ap_id or -1 if error
*            cc's : set by D0.W
*  changed:
*
*  This routine notifies GEM of an application and allows GEM to set up
*  some housekeeping.  An application ID is assigned and we store it
*  in our global variable area.
*
_appl_in:
        moveq    #APPL_INI,d1    ; code = initialize
        bsr.s    gem_ep          ; call GEM
        move.w   d0,ap_id(a5)    ; store application ID
        rts

* ============================
* ===== _APPL_EX : exit =====
* ============================
*
* in:   none
* out:  d0.w = 0 on error
*
*  This routine notifies GEM of our intention to terminate.
*
_appl_ex:
        moveq    #APPL_EXI,d1    ; code = exit
        bra.s    gem_ep          ; call GEM
* ================================
* ===== _OBJC_DR :  Object Draw =====
* ================================
*
* in:   d0.w = start obj
*       d1.w = depth
*       a0.l = address of x,y,w,h clip limits [4 words]
*       a3.l = tree address
* out:  d0.w = 0 if error occurred
*
_objc_dr:
        lea      int_in(a5),a1  ; address integer input array
        move.w   d0,(a1)+       ; store starting object
        move.w   d1,(a1)+       ; store drawing depth
        move.l   (a0)+,(a1)+    ; store x and y
        move.l   (a0)+,(a1)+    ; store w and h
        move.l   a3,addr_in(a5) ; store address of resource tree
        moveq    #OBJC_DRA,d1   ; set function code = object draw
        bra.s    gem_ep         ; call GEM

* ====================================
* ===== _FORM_DO :  Process Dialog =====
* ====================================
*
* in:   d0.w = start object, a3.l = tree address
* out:  d0.w = exit object
*
_form_do:
        move.l   a3,a0          ; pass tree address in A0
        moveq   #FORM_DO,d1     ; function = FORM DO
        bra.s    call_gem       ; call GEM

* ========================================
* ===== _FORM_DI : Dialog Housekeeping =====
* ========================================
*
* in:   d0.w = action [0=reserve, 1=grow, 2=shrink, 3=free]
*       a0.l = address of x,y,w,h limits [4 words]
* out:  d0.w = 0 if an error occurred
*
*  This routine is used to reserve or release memory or to draw a
*  growing or shrinking box.  The operation code is passed in D0 and
*  the second x,y,w,h dimensions are pointed to by a0.  This routine
*  is currently hard coded so that the first set of x,y,w,h points is
*  always zero.
```

```
*
_form_di:
        lea     int_in(a5),a1   ; address integer input array
        move.w  d0,(a1)+        ; store operation code
        clr.l   (a1)+           ; x1 = 0, y1 = 0
        clr.l   (a1)+           ; w1 = 0, h1 = 0
        move.l  (a0)+,(a1)+     ; store x2, y2
        move.l  (a0)+,(a1)+     ; store w2, h2
        moveq   #FORM_DIA,d1    ; function = FORM_DIALOG
        bra.s   gem_ep          ; call GEM

* ========================================
* ===== _FORM_AL :  Process Alert Boxes =====
* ========================================
*
* in:   d0.w = default exit button, a0 = alert string address
* out:  d0.w = exit button used, 1=first, 2=second, etc.
*
*  This routine displays and processes user interaction with the alert
*  box.  The address of the alert string is passed in A0 and the default
*  exit button if more than one is present is passed in D0.  The
*  alternate entry point '_alert' sets the default exit button to 1.
*
_alert:  moveq   #1,d0           ; default exit button = 1 (leftmost)
_form_al:
        moveq   #FORM_ALE,d1    ; function code
        bra.s   call_gem        ; call GEM

* ========================================
* ===== _FORM_CE : Calculate box centering =====
* ========================================
*
* in:   a3.l = address of tree
*       a1.l = addr of area to store x,y,w,h of centered tree
* out:  none
*
*  This routine calculates the x and y coordinates necessary to center
*  the resource treee addressed by A3 on the screen.  The x,y,w,h
*  coordinates necessary for a clip rectangle are returned and stored
*  at the address passed in A1.
*
_form_ce:
        move.l  a3,a0           ; pass tree address in A0
        moveq   #FORM_CEN,d1    ; function = form center
        bsr.s   call_gem        ; call gem
        lea     int_out+2(a5),a0 ; access x,y,w,h returned
        move.l  (a0)+,(a1)+     ; store x,y
        move.l  (a0)+,(a1)+     ; store w,h
        rts

* ======================================
* ===== _GRAF_MO : Change Mouse Form =====
* ======================================
*
*       in:     d0.w = code of mouse form desired
*      out:     d0.w = 0 if error
*  changed:
*
*  This routine changes the form of the mouse pointer used.  It can also
*  be used to hide and unhide the mouse.
*
_mouse:  moveq   #0,d0           set mouse form to pointer
_graf_mo:
        suba.l  a0,a0           moveq   #GRAF_MOU,d1

* ================================================
* ===== call_gem : set up parameters for a gem call =====
* ================================================
*
*       in:     D1.W = op code number
*               D0.W = optional int_in[0]
*               A0.L = optional addr_in[0]
*      out:     D0.W = returned value from int_out[0], if any
*  changed:     A0,A1,D1
*
```

continued

```
*  This subroutine uses the op code number passed in D1 to
*  look up the number of integers and addresses in and out for the
*  GEM call.  This information is then stored in the 'CONTROL' array
*  located in the dynamic storage area.  Then GEM call is then
*  performed and a possible return code is loaded into D0 before exit.
*
call_gem:
        move.l  a0,addr_in(a5)  in case there's an address in
        move.w  d0,int_in(a5)   in case there's an integer in

gem_ep: movem.l d2-d7/a1-a6,-(sp)       ; save registers
        add.w   d1,d1                   ; function x 2
        add.w   d1,d1                   ; function x 4
        lea     gemtbl(pc,d1.w),a0      ; form address of entry
*
* --- transfer opcode, integers in, integers out, and addresses
*
        lea     control(a5),a1  ; address control array
        moveq   #0,d0           ; form a zero
        movep.l d0,0(a1)        ; clear upper bytes of control[0..3]
        move.l  (a0),d0         ; get bytes from table entry
        movep.l d0,1(a1)        ; fill lower bytes of control[0..3]
        clr.w   8(a1)           ; set control[4] to zero
*
* --- perform the GEM/AES call
*
        move.l  a5,d1           ; pass address of 'control' array in D1
        move.w  #200,d0         ; pass special function number in D0
        trap    #2              ; call the system
        movem.l (sp)+,d2-d7/a1-a6 ; restore registers
        moveq   #0,d0           ; clear upper portion of register
        move.w  int_out(a5),d0  ; retrieve a return value
        rts
*
* --- Table of control parameters for AES calls
*
*     each entry has:  AES/GEM function number
*                      number of integers (words) in
*                      number of integers (words) out
*                      number of address (longs) in
*
gemtbl: dc.b    10,0,1,0        ; APPL_INIT
        dc.b    19,0,1,0        ; APPL_EXIT         dc.b    42,6,1,1        ; OBJC_DRAW
        dc.b    50,1,1,1        ; FORM_DO
        dc.b    51,9,1,0        ; FORM_DIAL
        dc.b    52,1,1,1        ; FORM_ALERT
        dc.b    54,0,5,1        ; FORM_CENTER
        dc.b    78,1,1,1        ; GRAF_MOUSE

*
* --- strings for alerts
*
warning: dc.b   '[3][ Clock missing or | not working ][ OK ]',0
badtime: dc.b   '[3][Bad Time or Date,|please retry][ OK ]',0
*
* --- strings for dialog
*
null:   dc.b    0

s_time: dc.b    '122345A',0
s1:     dc.b    'TIME = __:__:__ _M',0
s2:     dc.b    '999999F',0

s_date: dc.b    '071486',0
s4:     dc.b    'DATE = __/__/__',0
s5:     dc.b    '999999',0

s6:     dc.b    'SUN',0
s7:     dc.b    'MON',0
s8:     dc.b    'TUE',0
s9:     dc.b    'WED',0
s10:    dc.b    'THU',0
s11:    dc.b    'FRI',0
s12:    dc.b    'SAT',0
```

```
s13:    dc.b    'SET',0
s14:    dc.b    'EXIT',0
s15:    dc.b    'CLOCK CARTRIDGE',0
*
* ========== text info structures ========== *
*
*  each entry:   L - pointer to text
*                L - pointer to template
*                L - pointer to validation string
*                W - font to be used, 3=normal, 5=small
*                W - 6
*                W - justification (left, center, right)
*                W - color code
*                W - 0
*                W - border thickness ( + outward, - inward )
*                W - length of text (including null)
*                W - length of template (including null)
*               =====
*                28 bytes
*
*
* --- text info blocks*
ti0:    dc.l    s_time,s1,s2    ; time
        dc.w    3,6,0,$1180
        dc.w    0,-1,8,19

ti1:    dc.l    s_date,s4,s5    ; date
        dc.w    3,6,0,$1180
        dc.w    0,-1,7,16

ti2:    dc.l    s15,null,null   ; title string
        dc.w    3,6,2,$1180
        dc.w    0,-3,16,1
*
* ========== object structures ========== *
*
* each entry:  W - ID of next sibling (-1 if none)
*              W - ID of first offspring (-1 if none)
*              W - ID of last offspring (-1 if none)
*              W - type of object
*              W - object flags
*              W - state flags (selected, open, etc)
*              L - <expansion>
*              W - x position of upper left corner (relative to parent)
*              W - y position of upper left corner
*              W - width in pixels
*              W - height in pixels
*             =====
*              24 bytes
*
* --- object types:
*
BOX       EQU    20       ; box
TXT       EQU    21       ; text
BOXTEXT   EQU    22       ; text within a box
IMAGE     EQU    23       ; bit image
PROGDEF   EQU    24       ; programmer defined
IBOX      EQU    25       ; invisible box
BUTTON    EQU    26       ; text within a button
BOXCHAR   EQU    27       ; single char within a button
STRING    EQU    28       ; string
ETEXT     EQU    29       ; editable text
EBOXTEXT  EQU    30       ; editable text within a box
ICON      EQU    31       ; icon image
TITLE     EQU    32       ; string used in menu titles
*
* --- object option flags:
*
NONE      EQU    $0000    ; no option
SLECTBLE  EQU    $0001    ; SLECTBLE
DEFAULT   EQU    $0002    ; default for <CR>
EXIT      EQU    $0004    ; causes exit when selected
EDITABLE  EQU    $0008    ; editable text
RADIOB    EQU    $0010    ; radio button
LASTOB    EQU    $0020    ; last object in tree
```

continued

```
** --- object states:
*
NORMAL   EQU    $0000   ; nothing special
SELECTED EQU    $0001   ; has been selected by the mouse button
DISABLED EQU    $0008   ; can't be selected
OUTLINED EQU    $0010   ; shows up as outlined
SHADOWED EQU    $0020   ; shows up as casting a shadow
*
* ===== RESOURCE TREE:  ============================================
*
tree:   dc.w    -1,01,01,BOX            ; outermost box
        dc.w    NONE,NORMAL
        dc.l    $00011100
        dc.w    0,0,262,174
* obj 1
        dc.w    00,02,14,BOX
        dc.w    SLECTBLE+EXIT,NORMAL    ; inner framing box
        dc.l    $00FD1103
        dc.w    7,7,248,160
* obj 2
        dc.w    03,-1,-1,ETEXT          ; editable date string
        dc.w    EDITABLE,NORMAL
        dc.l    ti1
        dc.w    79,55,120,16
* obj 3
        dc.w    04,-1,-1,ETEXT          ; editable time string
        dc.w    EDITABLE,NORMAL
        dc.l    ti0
        dc.w    79,78,144,16
* obj 4
        dc.w    12,05,11,BOX            ; box surrounding radio buttons
        dc.w    NONE,NORMAL
        dc.l    $00FF1100
        dc.w    16,16,43,126
* obj 5
        dc.w    06,-1,-1,BUTTON         ; radio button for SUN
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s6
        dc.w    1,1,41,16
* obj 6
        dc.w    07,-1,-1,BUTTON         ; MON
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s7
        dc.w    1,19,41,16
* obj 7
        dc.w    08,-1,-1,BUTTON         ; TUE
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s8
        dc.w    1,37,41,16
* obj 8
        dc.w    09,-1,-1,BUTTON         ; WED
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s9
        dc.w    1,55,41,16
* obj 9        dc.w    10,-1,-1,BUTTON         ; THU
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s10
        dc.w    1,73,41,16
* obj 10
        dc.w    11,-1,-1,BUTTON         ; FRI
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s11
        dc.w    1,91,41,16
* obj 11
        dc.w    04,-1,-1,BUTTON         ; SAT
        dc.w    (RADIOB+SLECTBLE),NORMAL
        dc.l    s12
        dc.w    1,109,41,16
* obj 12
        dc.w    13,-1,-1,BUTTON         ; 'SET' button
        dc.w    5,SHADOWED
        dc.l    s13
        dc.w    72,112,64,16
```

```
* obj 13
        dc.w    14,-1,-1,BUTTON         ; 'EXIT' button
        dc.w    5,SHADOWED
        dc.l    s14
        dc.w    160,112,64,16
* obj 14
        dc.w    01,-1,-1,BOXTEXT        ; title in a box
        dc.w    LASTOB+SLECTBLE+EXIT,NORMAL
        dc.l    ti2
        dc.w    80,16,136,16
*
* --- dynamic storage area begins at end of program
*
*  The dynamic storage is an uninitialized temporary data area that
*  begins at the end of the program.  It is used for the stack and all
*  variables that do not need to be initialized prior to program
*  execution.  Use of a dynamic storage area reduces the disk space
*  requirements of the program.
*
storage: nop
                end
```

TIMESET.ASM Contributed by: Tim G. Hunkler
"68000 Machines: Atari 520ST Projects," Tim G. Hunkler. June, page 161.

```
******************
*               *
*  TIMESET.ASM  *  Set TIME from clock cartridge
*               *  <=> PUBLIC DOMAIN 09-SEP-86 <=>
******************
*
*** created:  07-JUN-86  by  Tim Hunkler / Solar Powered Software
*
* --- symbol definitions
*
ROM3    equ     $fa0000         ; address to strobe rom3 pin
ROM4    equ     $fb0000         ; address used to strobe rom4 pin
READ    equ     0               ; offset for RTC chip reads
WRT     equ     64              ; offset for RTC chip writes
CR      equ     $0d             ; carriage return
LF      equ     $0a             ; line feed
BELL    equ     $07             ; sound the bell

* ==============================
* ----- PROGRAM ENTRY POINT -----
* ==============================
*
x:      lea     x-16(pc),sp     ; set up a very small stack (240 bytes)
*
* --- initialize the clock cartridge after power up
*
        moveq   #%0001,d0       ; load control bits
        moveq   #0,d1           ; addr 0 = control register

        bsr     w_clock         ; enable clock, disable interrupts
*
* --- read the time from the cartridge
*
        movea.l sp,a5           ; point a5 to temp buffer
        bsr     r_clock         ; retrieve the time
        move.w  d0,-(sp)        ; save error flag
*
* --- set the system time
*
        bsr.s   set_tos         ; set the time if no error
*
* --- format and display the time
```

continued

```
*
        bsr     tod_fmt         ; format time for display
        lea     title(pc),a0    ; point to title string

        tst.w   (sp)+           ; error reading cartridge ?
        bge.s   okfine          ; no, skip
        lea     errmsg(pc),a0   ; yes, change message
okfine: bsr.s   string          ; display the string
*
* --- delay a bit before exiting
*
        move.l  #400000,d0      ; load delay count
delay:  subq.l  #1,d0           ; loop till count reaches zero
        bne.s   delay
*
* --- time to exit
*
        clr.w   -(sp)           ; pass function code = 0
        trap    #1              ; call the system (never return)
        page
* ======================================
* ===== string :  display a string =====
* ======================================
*
*       in:  A0.L = string address
*      out:  none
*  changed:  many
*
*  This subroutine displays the null terminated string whose address is
*  passed in A0 at the current cursor location on the screen.
*
string: move.l  a0,-(sp)        ; pass string address on stack
        move.w  #9,-(sp)        ; function 9 = text output
        trap    #1              ; system call
        addq.l  #6,sp           ; fix stack
        rts

* ======================================================
* ===== fetch2 : convert next two digits to a number =====
* ======================================================
*
*       in:  A0.L = address of digits
*      out:  D0.L = value of conversion
*            A0.L = updated by 2
*            cc's : set by D0
*  changed:  D1
*
*  This subroutine converts the two ASCII digits pointed to by the
*  address in A0 into a binary value.
*
fetch2: moveq   #%1111,d0       ; load mask
        and.b   (a0)+,d0        ; get 10's digit
        moveq   #%1111,d1       ; load mask
        and.b   (a0)+,d1        ; get 1's digit
        mulu    #10,d0          ; x 10
        add.w   d1,d0           ; result = d0 x 10 + d1
        rts

* ================================================
* ===== set_tos :  Set the time and date for TOS =====
* ================================================*
*       in:  A5.L = address of buffer returned by r_clock
*            D0.W = negative value if default time desired
*      out:  D7.L = date/time encoded like TOS likes it
*
*  changed:  A0,A1,A2,D1,D2
*
*  This subroutine uses the data retrieved from the clock cartridge and
*  stored in a formatted buffer and converts this information into a
*  format suitable for setting the time of both TOS and GEM.  System
*  calls are then performed for changing the time and date.
*
set_tos:
        move.l  #$0C215000,d7   ; preload   01/01/86 10:00:00
        tst.w   d0              ; do we want default or real time?
        bmi.s   stdef           ; branch for default
```

```
*
* --- format the date:  [yyyyyyy mmmm ddddd]
*
        lea     2(a5),a0        ; point to first year digit
        bsr.s   fetch2          ; get year (00= 1980)
        move.l  d0,d7           ; place year into d7

        bsr.s   fetch2          ; get month, 1..12
        lsl.w   #4,d7           ; form:  ????? yyyyyyy 0000
        or.w    d0,d7           ; add in the month

        bsr.s   fetch2          ; get day of month, 1..31
        lsl.w   #5,d7           ; form:  yyyyyyy mmmm 00000
        or.w    d0,d7           ; add in the day
*
* --- encode the time:  [hhhhh mmmmmm sssss]
*
        swap    d7              ; put year/month/day in upper word

        bsr.s   fetch2          ; get hour, 0..23
        move.w  d0,d7           ; form in low word:  ??????????? hhhhh

        bsr.s   fetch2          ; get minute, 0..59
        lsl.w   #6,d7           ; form:  ????? hhhhh 000000
        or.w    d0,d7           ; add in the minutes

        bsr.s   fetch2          ; get seconds, 0..59
        lsl.w   #5,d7           ; form:  hhhhh mmmmmm 00000
        lsr.w   #1,d0           ; divide seconds by 2
        addx.w  d0,d7           ; add in seconds (with rounding)
*
* --- set the TOS time  [date : time ]
*
stdef:  move.w  d7,-(sp)        ; pass the new time
        move.w  #45,-(sp)       ; pass function code = set time
        trap    #1              ; call system
        addq.l  #4,sp           ; fix stack
*
* --- set the GEM date and time:    [ date : time ]*
        move.l  d7,-(sp)        ; pass the date and time
        move.w  #22,-(sp)       ; pass function code = set date and time
        trap    #14             ; call system
        addq.l  #6,sp           ; fix stack
*
* --- set the TOS date
*
        swap    d7              ; retrieve date to low word
        move.w  d7,-(sp)        ; pass the date
        move.w  #43,-(sp)       ; pass function code = set date
        trap    #1              ; call system
        addq.l  #4,sp           ; fix stack
        rts

* ====================================
* ===== w_clock :  write to clock =====
* ====================================
*
*       in:  D0.B = data to be written
*            D1.W = register address x 2 to be written to
*
*      out:  A3.L = pointer to clock address latch
*            A4.L = pointer to clock chip select
*            D1.W = increased by 2
*            D0.W = data present prior to write
*            cc's : set by D0
*
*  changed:  A0,D2
*
*  This subroutine is used to write data to the clock chip.  The data
*  to be written is passed in the lower 4 bits of D0 and 2 times the
*  clock register to be accessed in passed in D1.
*
w_clock:
        lea     ROM3,a3         ; LATCH address
        lea     ROM4,a4         ; CS low strobe
```

continued

```
*
* --- form the clock register address
*
wcep:   lea     0(a3,d1.w),a0   ; form proper address
*
* --- mask off 4 bit data and align it into address bits 12..15
*
        moveq   #15,d2          ; load mask
        and.w   d0,d2           ; mask of 4 bit data item
        ror.w   #4,d2           ; rotate to upper bits
*
* --- latch address, data, and enable writes to the chip
*
        move.b  WRT(a0,d2.l),d0 ; read old data, latch new data
*
* --- turn chip select on to start the write operation
*
        tst.w   (a4)            ; set chip select low
        addq.w  #2,d1           ; update register address
*
* --- turn off write enable, turn off chip select, but hold data
*
        tst.b   READ(a0,d2.l)   ; set chip select high
        andi.w  #$000F,d0       ; mask off 4 bit data and set cond codes
        rts

* ====================================
* ===== r_clock : read from clock =====
* ====================================
*
*       in:  A3.L = pointer to clock latch address
*            A4.L = pointer to chip select strobe address
*            A5.L = address of 16 byte buffer to be filled
*
*      out:  D0.L = 0 if all OK, else -1
*            cc's : set by D0
*            (A5) ---> [ flag bits    ]  leap count, AM/PM, 12/24 hour
*                 +1   [ day of week  ]  1=Sunday
*                 +2   [ year x 10    ]  00 = 1980, 01 = 1981, etc.
*                 +3   [ year x 1     ]
*                 +4   [ month x 10   ]
*                 +5   [ month x 1    ]
*                      [ day x 10     ]
*                      [ day x 1      ]
*                      [ hour x 10    ]
*                      [ hour x 1     ]
*                      [ minute x 10  ]
*                      [ minute x 1   ]
*                      [ second x 10  ]
*                      [ second x 1   ]
*                +14   [ .1 seconds   ]
*                +15   [ junk byte    ]
*
*  changed:  A0,D1,D2
*
*  This subroutine reads all time and date registers from the clock chip
*  and fills a 16 byte buffer whose address was passed in A5.  As the
*  information is read it is added up.  Invalid sums typically indicate
*  a dead battery or a clock cartridge which is not present and are
*  indicated by returning a -1 in register D0 and the condition codes.
*  A data changed bit is check and if the time changed while the read
*  operation was in progress the operation is repeated.
*
r_clock:
        movea.l a3,a0           ; set pntr to register 0
        lea     16(a5),a5       ; point to end of buffer to fill
        moveq   #0,d1           ; set accumulator to zero
*
* --- preload the register address in the latch
*
        tst.w   (a0)+           ; set register selection to 0
*
* --- loop and read clock registers 0..15*
        moveq   #15,d2          ; loop count = 16
```

```
fetch:  tst.w    (a4)           ; set chip select on
        moveq    #15,d0         ; load 4 bit data mask
        and.w    (a0)+,d0       ; get data, set next addr, select off
        move.b   d0,-(a5)       ; store data in buffer
        add.w    d0,d1          ; add up all data values
        dbf      d2,fetch       ; and repeat for 16 items
*
* --- retrieve data changed flag (zero indicates no change occurred)
*
        tst.w    (a4)           ; set CS low with register select = 0
        moveq    #8,d0          ; load mask
        and.w    (a0),d0        ; get flag, nxt addr = 2, select = off
*
* --- if no clock present we generally accumulate 16x15 = 240
*
        cmpi.w   #125,d1        ; does accumulate indicate bad data?
        ble.s    nwflg          ; no, branch
        moveq    #-1,d0         ; yes, set error flag
*
* --- test data changed flag and reread the time if necessary
*
nwflg:  tst.w    d0             ; was data changed (or an error)?
        bgt.s    r_clock        ; yes, then repeat the time reading
        rts                     ; no, exit with cond codes set

* ========================================================
* ===== xlate : Translate two bytes into ASCII digits =====
* ========================================================
*
*       in:  A0.L = address of two bytes in the range 0..9
*            A1.L = destination address for ASCII digits
*      out:  A0.L = incremented by 2
*            A1.L = incremented by 3
*  changed:  none
*
*  This subroutine converts two bytes addressed by A1 into ASCII digits
*  in the range '0..9' and increments the address pointer by 3.
*
xlate:  bsr.s    dt             ; convert digit one
        bsr.s    dt             ; convert digit two
        addq.l   #1,a1          ; skip ahead one place
        rts

dt:     move.b   (a0)+,(a1)     ; copy byte
        addi.b   #$30,(a1)+     ; convert copied byte to ASCII
        rts

* ========================================
* ===== B2D :  Binary to Decimal String =====
* ========================================
*
*       in:  D0.W = value to convert, range = 0..99
*            A1.L = destination buffer*      out:  A1.L = incremented by 2
*  changed:  D0
*
*  This subroutine converts the value passed in D0.W which should be in
*  the range of 0..99 into a two digit ASCII string and stores these
*  digits at the address passed in A1.
*
b2d:    ext.l    d0             ; extend word into longword
        divu     #10,d0         ; divide to separte ten's and one's
        addi.b   #$30,d0        ; convert ten's digit to ASCII
        move.b   d0,(a1)+       ; store ten's digit
        swap     d0             ; swap remainder to low word
        addi.b   #$30,d0        ; convert one's digit to ASCII
        move.b   d0,(a1)+       ; store one's digit
        rts

* ================================================
* ===== tod_fmt :  format time of day and date =====
* ================================================
*
*       in:  A5.L = pointer to retrieved time buffer
*      out:  none
*  changed:  A0,A1,D0,D1,?
```

continued

```
*
*  This subroutine takes the data block returned by the subroutine
*  'r_clock' and formats it into a string of the form:
*
*      "WED  07-SEP-86  12:42 AM"
*
tod_fmt:
        lea     time+3(pc),a1   ; address day of week field
        lea      1(a5),a0       ; pnt to day of week byte

        moveq   #0,d0           ; ensure we start with zero
        move.b  (a0)+,d0        ; get day of week, 1..7
        lsl.w   #2,d0           ; form 4x which is byte offset
        move.l  t_day-4(pc,d0.w),(a1)+ ; install string like "MON "
        addq.l  #8,a1           ; address year field
        bsr     fetch2          ; fetch the year, 0..99
        addi.w  #80,d0          ; convert 0 into 80
        bsr.s   b2d             ; format as decimal string

        lea     time+11(pc),a1  ; address month field
        bsr     fetch2          ; fetch the month, 1..12
        lsl.w   #2,d0           ; form 4x which is byte offset
        move.l  month-4(pc,d0.w),(a1) ; install string like "JAN-"

        subq.l  #3,a1           ; address day of month
        bsr.s   xlate           ; copy and translate day of the month

        addq.l  #8,a1           ; address hours field
        bsr     fetch2          ; fetch hours, 0..23
        moveq   #$41,d2         ; preload "A" for AM
        tst.w   d0              ; is time 00:xx ?
        bne.s   ampm            ; no, skip
        moveq   #12,d0          ; yes, change 00:xx into 12:xx
        bra.s   AM

ampm:   cmpi.w  #12,d0          ; are we after noon?
        blt.s   AM              ; no, skip
        beq.s   NOON            ; if before 1:00pm just update character
        subi.w  #12,d0          ; convert 13:00 into 1:00
NOON:   moveq   #$50,d2         ; change character to "P"

AM:     bsr.s   b2d             ; format hours as decimal string
        addq.l  #1,a1           ; advance over colon
        bsr.s   xlate           ; minutes
        bsr.s   xlate           ; seconds

        move.b  d2,(a1)         ; install AM or PM
        rts
*
* --- This is a table for converting 1..7 into a day of week string
*
t_day:  dc.l    'Sun ','Mon ','Tue ','Wed '
        dc.l    'Thu ','Fri ','Sat '
*
* --- This is a table for converting 1..12 into a month string
*
month:  dc.l    'Jan-','Feb-','Mar-','Apr-'
        dc.l    'May-','Jun-','Jul-','Aug-'
        dc.l    'Sep-','Oct-','Nov-','Dec-'
*
* --- the title and time and date display
*
title:  dc.b    CR,LF,LF
        dc.b    '=TIMESET v1.0='
time:   dc.b    CR,LF
        dc.b    ' WED  xx-xxx-xx  xx:xx:xx xM'
        dc.b    CR,LF,0
*
* --- error message
*
errmsg: dc.b    CR,LF,BELL
        dc.b    '=TIMESET='
        dc.b    CR,LF
```

```
        dc.b    ' * Check Clock Battery!'
        dc.b    CR,LF,0

        end
```

COMPRES.PAS Contributed by: Dick Pountain
"Focus on Algorithms: Run-Length Encoding," by Dick Pountain. June 1987, page 317.

```
{COMPRESS.PAS is a procedure written in Turbo Pascal for the IBM PC
 and its compatibles for the purpose of compressing screen data.
 It is not a stand-alone program.}

procedure Compress;

const escapechar = $F800;
       scrnseg  = $B800;
       scrnsize = 4000;

var OutputFile: Text;
    OutputFileName: string[80];
    runlength,currentword,nextword,scrnofs,items: integer;

begin
  OutputFileName := paramSTR(1);
  Assign(OutputFile, OutputFileName);
  Rewrite(OutputFile);
  write(OutputFile,'(');
  items := 0;
  scrnofs := 0;
  currentword := MemW[scrnseg:scrnofs];   {read word from screen memory}
  scrnofs := scrnofs + 2;
  repeat
    runlength := 0;
    repeat
      nextword := MemW[scrnseg:scrnofs];
      scrnofs := scrnofs + 2;
      runlength := runlength + 1;
    until (nextword <> currentword) or (scrnofs > scrnsize);
    if runlength > 1
    then begin                                      {it's count/value}
      runlength := escapechar or runlength;         {set 'escape' bits}
      write(OutputFile,runlength,',',currentword,',');
      if (items mod 12) >= 10                       {format into lines}
        then writeln(OutputFile);
      items := items + 2
    end
    else begin                                      {it's a singleton}
      write(OutputFile,currentword,',');
      if (items mod 12) >= 11
        then writeln(OutputFile);
      items := items + 1
    end;
    currentword := nextword
  until scrnofs > scrnsize;
  write(OutputFile,'*** ',items,' items ***);');
  Close(OutputFile);  writeln('Compressed data written to ',OutputFileName)
end;
```

EXPAND.PAS   Contributed by: Dick Pountain
"Focus on Algorithms: Run-Length Encoding," by Dick Pountain. June 1987, page 317.

```
{EXPAND.PAS is a procedure written for the IBM PC and its compatibles
```

continued

```
 in Turbo Pascal for the purpose of expanding data compressed with
 COMPRESS.PAS.  It is not a stand-alone program.}

{Fill <count> words of memory starting at <seg:ofs>
 with the 16-bit value <word>}

procedure FillW(seg,ofs,count,word:integer);
begin
    inline
    ($8B/$86/seg/        {MOV AX,seg}
     $8E/$C0/            {MOV ES,AX}
     $8B/$BE/ofs/        {MOV DI,ofs}
     $8B/$86/word/       {MOV AX,word}
     $8B/$8E/count/      {MOV CX,count}
     $FC/                {CLD}
     $F3/$AB)            {REPZ STOSW}
end;

procedure Expand(srcofs,picsize:integer);

const escapechar = $F800;       {binary 1111100000000000}
      transparent = $07FA;
      scrnseg = $B800;          {start segment of video RAM}

var srcptr,destptr,data,runlength,i: integer;

begin
  srcptr  := 0;
  destptr := 0;
  while srcptr < picsize * 2 do
  begin
    data   := MemW[Cseg:srcofs+srcptr];            {fetch next word}
    srcptr := srcptr + 2;
    if (data and escapechar) = escapechar          {test top 5 bits}
    then begin                                     {it's a count word}
      runlength := data xor escapechar;            {unpack count part}
      data  := MemW[Cseg:srcofs+srcptr];           {fetch next word}
      srcptr := srcptr + 2;
      if data = transparent                        {color is transparent}
      then destptr := destptr + (2 * runlength)    {so just bump pointer}
      else begin
        FillW(scrnseg,destptr,runlength,data);     {fill screen memory}
        destptr := destptr + (2 * runlength)
      end
    end
    else begin                                     {it's a singleton}
      MemW[scrnseg:destptr] := data;       destptr := destptr + 2
    end
  end
end;
```

POLYFIT.BAS Contributed by: William G. Hood
Programming Insight: "Polynomial Curve Fitter," William G. Hood. June 1987, page 155.

```
1000 LN=1000: LD=11: REM LN=Max data points; LD=highest degree+1
1010 DEF FNMI(X,Y)=(X<Y)*(-X) + (Y<=X)*(-Y)
1020 N=0: M=0: S2=0: R2=0: MF=0
1030 S1=0: S2=0: S3=0: S4=0: P1=0: P2=0: P3=0: I=0: J=0: J1=0: K=0: VR=0:MM=0: WT=0: P=0
1040 DIM X(LN),Y(LN),W(LN),C(LD)
1050 DIM D1(LD),D2(LD),D3(LD),D4(LD),D5(LD),D6(LD)
1060 GOTO 1450
1070 IF MF>0 AND M>MM THEN J1=MM+1: MM=M: GOTO 1130
1080 J1=1: MM=M: S1=0: S2=0: S3=0: S4=0
1090 FOR I=1 TO N: WT=W(I)
1100 S1=S1+WT*X(I): S2=S2+WT: S3=S3+WT*Y(I): S4=S4+WT*Y(I)*Y(I)
1110 NEXT I
1120 D4(1)=S1/S2: D5(1)=0: D6(1)=S3/S2: D1(1)=0: D2(1)=1: VR=S4-S3*D6(1)
1130 FOR J=J1 TO MM: S1=0: S2=0: S3=0: S4=0
1140 FOR I=1 TO N: P1=0: P2=1
```

```
1150 FOR K=1 TO J: P=P2: P2=(X(I)-D4(K))*P2-D5(K)*P1: P1=P: NEXT K
1160 WT=W(I): P=WT*P2*P2
1170 S1=S1+P*X(I): S2=S2+P: S3=S3+WT*P1*P1: S4=S4+WT*Y(I)*P2: NEXT I
1180 D4(J+1)=S1/S2: D5(J+1)=S2/S3: D6(J+1)=S4/S2: D3(1)=-D4(J)*D2(1)-D5(J)*D1(1)
1190 IF J<4 THEN 1210
1200 FOR K=2 TO J-2: D3(K)=D2(K-1)-D4(J)*D2(K)-D5(J)*D1(K): NEXT K
1210 IF J>2 THEN D3(J-1)=D2(J-2)-D4(J)*D2(J-1)-D5(J)
1220 IF J>1 THEN D3(J)=D2(J-1)-D4(J)
1230 FOR K=1 TO J: D1(K)=D2(K): D2(K)=D3(K): D6(K)=D6(K)+D3(K)*D6(J+1): NEXT K
1240 NEXT J
1250 FOR J=1 TO M+1: C(J)=D6(M+2-J): NEXT J
1260 P2=0: FOR I=1 TO N: P=C(1)
1270 FOR J=1 TO M: P=P*X(I)+C(J+1): NEXT J
1280 P=P-Y(I): P2=P2+W(I)*P*P: NEXT I
1290 S2=0: IF N>M+1 THEN S2=P2/(N-M-1)
1300 R2=1: IF VR<>0 THEN R2=1-P2/VR: IF R2<0 THEN R2=0
1310 RETURN
1320 REM GOSUB 30 calls the subroutine
1330 REM Input:
1340 REM N=# of data points
1350 REM X()=X-coordinates of the data points
1360 REM Y()=Y-coordinates of the data points
1370 REM W()=Weighting factors of the data points
1380 REM M=Degree of polynomial
1390 REM MF=0 if new data, MF=1 if old data but higher degree
1400 REM
1410 REM Output:
1420 REM C=Array of M+1 coefficients
1430 REM S2=Residual variance
1440 REM R2=coefficient of determination
1450 CLS1460 PRINT "Polyfit. Copyright (C) 1986 by William G. Hood"
1470 PRINT
1480 PRINT "This program finds the coefficients of the nth degree polynomial"
1490 PRINT: PRINT "y = c(1)*x^n + c(2)*x^(n-1) + ... + c(n)*x + c(n+1)
1500 PRINT: PRINT "that fits a set of data points in a least-squares sense."
1510 PRINT "Each data point must consist of an X value, a Y value, and an
1520 PRINT "optional weight, separated by commas and terminated by a return.
1530 PRINT "Data are read from a disk file until the eof is reached, or a
1540 PRINT "specified number of data points are read in from the keyboard."
1550 PRINT
1560 LINE INPUT "Name the data file (null line=keyboard): "; FI$
1570 PRINT "Is the data weighted (Y/N, null line=No)? ";
1580 W$="": INPUT W$
1590 IF W$<>"Y" AND W$<>"y" THEN W$="N"
1600 IF FI$<>NU$ THEN 2000
1610 PRINT "Keyboard data entry"
1620 PRINT "How many data points ( 2 <= n <=";LN;")";: INPUT N
1630 IF N<2 OR N>LN THEN 1620
1640 FOR K=1 TO N
1650 IF W$<>"N" THEN INPUT "x,y,w";X(K),Y(K),W(K): W(K)=ABS(W(K))
1660 IF W$="N" THEN INPUT "x,y"; X(K),Y(K): W(K)=1
1670 NEXT K
1680 PM=FNMI(LD-1,N-1)
1690 PRINT "Degree of polynomial ( 1<= m <= "; PM;")";: INPUT M: M=INT(M)
1700 IF M<1 OR M>PM THEN 1690
1710 GOSUB 1070
1720 PRINT: PRINT "Coefficients (constant term last): ": K=0
1730 FOR J=1 TO M+1: PRINT TAB(K) C(J);: K=K+20: IF K>20 THEN K=0: PRINT
1740 NEXT J
1750 PRINT: PRINT "Residual variance: ";S2
1760 R2=INT(1000000!*R2+.5)/1000000!
1770 PRINT: PRINT"Coefficient of determination: ";R2
1780 PRINT "Try another degree (Y/N, null line=No)";: A$="": INPUT A$
1790 IF A$="Y" OR A$="y" THEN 1680
1800 PRINT "Print a table of the data (Y/N, null line=No)";: A$="": INPUT A$
1810 IF A$<>"Y" AND A$<>"y" THEN 1870
1820 PRINT: PRINT"Degree of p(x): ";M
1830 PRINT: PRINT " X"; TAB(11) "Y"; TAB(25) "p(x)": PRINT
1840 FOR I=1 TO N: P=C(1)
1850 FOR K=1 TO M: P=P*X(I)+C(K+1): NEXT K
1860 PRINT X(I);TAB(10);Y(I);TAB(24);P: NEXT I
1870 IF F$<>NU$ THEN END
1880 PRINT: PRINT "Save the data (Y/N, null line=No)? ";: A$="": INPUT A$
1890 IF A$<>"Y" AND A$<>"y" THEN END
1900 PRINT: LINE INPUT "Name of output file ";FO$
```

continued

```
1910 IF FO$=NU$ THEN END
1920 PRINT "Writing to ";FO$
1930 OPEN FO$ FOR OUTPUT AS 1
1940 FOR I=1 TO N
1950 IF W$<>"N" THEN PRINT#1, X(I);","; Y(I);","; W(I)
1960 IF W$="N" THEN PRINT#1, X(I);","; Y(I)
1970 NEXT I
1980 CLOSE 1
1990 END
2000 PRINT "Reading from ";FI$2010 OPEN FI$ FOR INPUT AS 1
2020 N=0
2030 PRINT "  X" TAB(15) "Y"; TAB(28) "W"; : PRINT
2040 IF EOF(1) OR N=LN THEN 2100
2050 N=N+1
2060 IF W$<>"N" THEN INPUT#1, X(N), Y(N), W(N): W(N)=ABS(W(N))
2070 IF W$="N" THEN INPUT#1, X(N), Y(N): W(N)=1
2080 PRINT X(N) TAB(14) Y(N) TAB(28) W(N)
2090 GOTO 2040
2100 CLOSE 1
2110 PRINT: PRINT"File contained "; N;" data points."
2120 IF N<2 THEN PRINT "Too few data points.": END
2130 GOTO 1680
```

```
BENCH.OBJ  Contributed by:  Mat Davis
TEXT "Smalltalk/V Release 1.2," by Mat Davis, June, page 256.
```

```
"Class implementing the Byte benchmarks.

 Written by Mat Davis."

Object subclass: #Benchmark
  instanceVariableNames: ''
  classVariableNames: ''
  poolDictionaries: '' !

!Benchmark class methods ! !


!Benchmark methods !

calculate: nTimes
        "Perform the Byte calculation benchmark.  Using 'print it' with
         the expression 'Benchmark new calculate: 5000' will print the
         error after 5000 iterations."
    | a b c |
    a := 271828 / 100000.
    b := 314159 / 100000.
    c := 1.
    1 to: nTimes do: [ :i |
        c := c  a.
        c := c  b.
        c := c / a.
        c := c / b ].
    ^c - 1!

diskRead: kBytes
        "Perform the Byte disk read benchmark.  Evaluating the expression
         'Benchmark new diskRead: 64' will read a 64K from the file
         A:TEST.DAT."
    | nr input |
    nr := kBytes  8.
    input := DiskA file: 'TEST.DAT'.
    1 to: nr do: [ :i |
        input next: 128 ].
    input close.
    ^'Done'!
```

```
diskWrite: kBytes
        "Perform the Byte disk write benchmark.  Evaluating the expression
         'Benchmark new diskWrite: 64' will create a 64K file named
         A:TEST.DAT."
    | a b nr output |
    a := '12345678123456781234567812345678'.
    b := a, a, a, a.    nr := kBytes  8.
    output := DiskA file: 'TEST.DAT'.
    1 to: nr do: [ :i |
        output nextPutAll: b ].
    output close.
    ^'Done'!

sieve: size
        "Perform the Byte prime sieve benchmark.  Evaluating the expression
         'Benchmark new sieve: 8191' will return the number of primes from
         1 to 8191."
    | flags count k |
    flags := Array new: size.
    count := 0.
    1 to: size do: [ :i | flags at: i put: 1 ].
    2 to: size do: [ :i |
        (flags at: i) == 1 ifTrue: [
            k := i + i.
            [ k > size ] whileFalse: [
                flags at: k put: 0.
                k := k + i ].
            count := count + 1 ] ].
    ^count! !
```

BYTE

1986 EDITORIAL INDEX

12-bit 16-channel high-speed analog-to-digital converter, construction of (S. Ciarcia), Jan 105

286i computer from Kaypro, review (H. Krause), Mar 217

520ST computer from Atari. *See* Atari computers, 520ST

1040ST computer from Atari compared to Macintosh and Amiga computers (B. Webster), May 343
preview (P. Robinson, J. R. Edwards), Mar 84

1409 EPROM programmer from B&C Microsystems, review (R. Jacobs), May 279

6060 computer from Xerox, review (W. Rash Jr.), Sept 275

8031 processor from Intel, construction of emulator board for in-circuit emulation of (G. Dinwiddie), July 181

65C816 processor in Apple IIGS computer, Oct 84

68000 processor from Motorola. *See* Motorola, 68000 processor

68008 processor from Motorola, compared to other 68000-series processors (T. L. Johnson), Sept 205

68010 processor from Motorola. *See* Motorola, 68010 processor

68012 processor from Motorola, compared to other 68000-series processors (T. L. Johnson), Sept 205

68020 processor from Motorola
compared to other 68000-series processors (T. L. Johnson), Sept 205
in Definicon 68020 coprocessor (T. Marshall, C. Jones, S. Kluger), July 120, Aug 108

68881 floating-point chip from Motorola, in Definicon 68020 coprocessor, July 120

80286 processor from Intel. *See* Intel, 80286 processor

80287 processor from Intel, in micro-based supercomputer (N. H. Christ, A. E. Terrano), Apr 145

80386 processor from Intel. *See* Intel, 80386 processor

A

A-200 Canon computer, review (P. V. Callamaras), Jan 293

Abstract mathematical art (K. E. Perry), Dec 181

Abundance Forth-based database language, in public domain (R. Green), Oct 193

Accelerator boards for IBM PC (S. S. Fried), IBM 141

Access Associates Alegra Memory Expansion Box for Amiga (B. Webster), Dec 305

ACE 2200 computer from Franklin, review (A. S. Woodhull), Sept 263

Acorn Computers RISC processor (D. Pountain), Jan 387

Acta outline processor for Macintosh (B. Webster), Dec 316

Activation network erector set (J. Pollack), Feb 196

Actor object-oriented language (C. B. Duff), Aug 211

Ada, best of BIX on, Nov 405

Addison-Wesley MicroTEX typesetting package, review (H. R. Varian), Apr 267

Advanced Fullscreen Debug debugging program, review (J. C. Carden), Apr 249

Advanced Trace86 assembler/debugger (B. Webster), Oct 293

Advantage! multifunction board for IBM PC AT, review (T. J. Byers), Jan 327

Aegis Animator and Images animation and painting programs, review (W. Block), Nov 285

Aging, robotic aids increasing independence in (K. G. Engelhardt, R. Edwards), Mar 191

Airus AI: Typist word processor (E. Shapiro), July 383

Aitken extrapolation for approximating integrals (D. M. Smith), Dec 113

Alegra Memory expansion Box for Amiga (B. Webster), Dec 305

Algorithms
Euclid's (R. T. Kurosaka), Jan 397, Mar 343
for extracting *n*th root from binary number (L. S. Wo), Nov 115
Huffman, for data compression (J. Amsterdam), May 99
ID3, for extracting knowledge from data (B. Thompson, W. Thompson), Nov 149
Pan and Reif, for inversion of large matrices (T. E. Phipps Jr.), Apr 181
Runge-Kutta, for solving differential equations, Apr 216
in BASIC (D. M. Leo), Apr 196
in FORTRAN, (B. Thomas), Apr 191
on storage allocators, Oct 125, 128, 130

Allocating memory in Modula-2, programming project (J. Amsterdam), Oct 123

Alloy Computer Products FT-60 tape backup unit, review (M. C. Rubel), Oct 243

Alpha Microsystems Videotrax tape backup unit, review (M. C. Rubel), Oct 243

American National Standards Institute (ANSI)
C language standards of (S. A. Hersee, D. Knopoff), Mar 135
meeting schedule of, Mar 136

Amiga computer from Commodore
Access Associates Alegra Memory Expansion Box for (B. Webster), Dec 305
Aegis Animator and Images animation and painting programs for, review (W. Block), Nov 285
AmigaDOS operating system of (D. Pountain), Feb 321
animation of graphics with Aegis Animator package (W. Block), Nov 285
glossary on, Sept 241
software for (E. A. Ditton, R. A. Ditton), Sept 241
best of BIX on, Feb 363, Mar 367, Apr 349, May 398, June 367, July 393, Aug 344, Sept 380, Oct 324, Nov 385, Dec 349
compared to Macintosh and Atari computers (B. Webster), Mar 305, May 343
Deluxe Paint program for, Apr 322
Electronic Arts software for (B. Webster), Apr 322, Dec 308
Instant Music program for (B. Webster), Dec 308
Lattice C compiler for, review (C. Heath), Nov 271
Lattice Screen Editor program editor for (B. Webster), July 352
Mandelbrot program for, in Lattice C. (P. B. Schroeder), Dec 207
Manx Aztec C68K C compiler for, review (C. Heath), Nov 271
Metascope debugger for (B. Webster), Nov 334
MicroBotics MAS-20 hard disk drive for (B. Webster), Dec 306
review on Model 1000 (T. Thompson), Oct 231
ROM Kernel software of, introduction to (R. J. Mical), Feb 116
sound system of (D. D. Thiel), Oct 139
system calls of, compared to Macintosh (A. B. Webber), Sept 249
three-dimensional Breakout game for (B. Webster), Dec 310
TxEd program editor for (B. Webster), Nov 336

AmigaDOS operating system, development of (D. Pountain), Feb 321

Amsoft Amstrad PCW 8256 computer and word processor (D. Pountain), Mar 333

Amstrad PCW 8256
computer and word processor (D. Pountain), Mar 333

Anagram solving in Pascal (B. Keefer), July 113

Analog circuit analysis
introduction to (W. Blume), July 165
with SPICE program for Commodore 64 (D. McNeill), July 170

Analog-to-digital converter, construction of (S. Ciarcia), Jan 105

Analytical software
Reflex database program, review (R. DeMaria), Aug 277
SPSS/PC+ package, review (J. M. Jacques), Nov 279

Anamartic Wisper wafer (D. Pountain), Nov 351

Animal Game in Smalltalk, Aug 153

Animation of Amiga graphics
with Aegis Animator package, review (W. Block), Nov 285
glossary on, Sept 246
software for (E. A. Ditton, R. A. Ditton), Sept 241

Animator animation software for Amiga, review (W. Block), Nov 285

Ann Arbor Softworks
FullPaint program for Macintosh (E. Shapiro), Sept 368

Ansa Software
Application Generator from, Sept 308
Paradox 1.1 relational database program from, review (R. DeMaria), Sept 303
Runtime module from, Sept 308

ANSI (American National Standards Institute)
C language standards of (S. A. Hersee, D. Knopoff), Mar 135
meeting schedule of, Mar 136

ANSI.SYS driver,
Enhanced Console Driver replacement for (A. Zackin), Oct 183

Apex Resources
DevpacST 68000 assembly language development package (B. Webster), Nov 330

APL language, review on Pocket APL version (E. H. Johnson), Mar 237

Apple computers
Apple II
best of BIX on, Nov 394, Dec 370
for blind users, Mar 199, 251
Franklin ACE 2200 computer compared to (A. S. Woodhull), Sept 263
MIDI interfaces for, hardware review (R. Powell, R. Grehan), June 265
Printit printer interface card for, review (H. Brugsch, J. J. Lazzaro), Mar 261
Sider hard disk drive for, review (D. E. Hall), Jan 319
text analysis programs identifying unknown authors for (J. Tankard), Feb 231
three-dimensional graphics program for, in BASIC, Jan 153
windowing system for, programming of (B. Webster), Mar 128, Apr 97
Apple IIGS
best of BIX on, Dec 376
preview (G. Williams, R. Grehan), Oct 84
catalog sort routine for ProDOS of (A. C. Silvestri), June 117
LaserWriter Plus laser printer from, June 88
MacApp object-oriented application framework for Macintosh from (K. J. Schmucker), Aug 189
programming experience with, Aug 200
MacDraw painting program from, review (R. Birmele), May 269
Macintosh. *See* Macintosh computer
object-oriented languages for Macintosh from, Aug 177

Approximations in computer computations
with Aitken extrapolation for approximating integrals (D. M. Smith), Dec 113
Chebyshev, Apr 174
in Runge-Kutta method for solving differential equations, Apr 216
in BASIC (D. M. Leo), Apr 191
in FORTRAN (B. Thomas), Apr 191
speed and precision of (S. L. Moshier), Apr 161

Apricot XEN computer (D. Pountain), Apr 305

Arabic script typesetting (P. A. MacKay), Feb 201

Architecture of houses, bit-mapped classifier system on, in BASIC, Nov 166

Arity/Prolog
version 3.2, review (W. G. Wong), Mar 245
version 4.0, Mar 247

Arms, robotic
in automation of chemistry laboratory (G. W. Kramer, P. L. Fuchs), Jan 263
coordination of multiple manipulators (J. S. Hawker, R. N. Nagel, R. Roberts, N. G. Odrey), Jan 203

Artificial intelligence
in computer vision (J. L. Cuadrado, C. Y. Cuadrado), Jan 237
knowledge representation in. *See* Knowledge representation

Ashton-Tate
dBASE III Plus software from (E. Shapiro), June 330
Framework II software from, Apr 331

Assembly language
best of BIX on, IBM 300
compared to C language (T. Hogan), IBM 267
Getspec routine for passing filenames to compiled BASIC in (B. Hubanks), Nov 119
guidelines for Motorola MC68000 processor (M. Morton), Sept 163
HiSoft DevpacST 68000 assembly language development package, Nov 330
interrupt routines for IBM PC in (W. J. Claff), IBM 249
using assembly routines in MS-FORTRAN programs (M. Dahmke), IBM 217

AST Research
Advantage! multifunction board from, review (T. J. Byers), Jan 327
Enhanced Expanded Memory Specification from, IBM 170

ASYST laboratory interfacing software package, July 303

Atari computers
520ST (B. Webster), Feb 331
compared to Macintosh and Amiga computers (B. Webster), Mar 305, May 343
DEGAS painting software for (B. Webster), Apr 320
description of (J. R. Edwards, P. Robinson, B. McLaughlin), Jan 84
MichTron software for (B. Webster), Apr 318
review (E. Jensen), June 233
1040ST
compared to Macintosh and Amiga computers (B. Webster), May 343
preview (P. Robinson, J. R. Edwards), Mar 84
best of BIX on, Feb 372, Mar 376, Apr 366, May 402, June 372, July 397, Aug 347, Sept 390, Oct 328, Nov 386, Dec 356
games for (B. Webster), Dec 312
Graphics Environment Manager of, Sept 223
Line A Handler of, Sept 223
Megamax C development system for (B. Webster), Nov 326
Motorola 68000 processor and TOS operating system of (M. Rothman), Sept 223
Personal Pascal software for (B. Webster), July 347
Shivji interview on (P. Robinson), Mar 90
XBIOS of, Sept 223

ATOMCC toolbox for solving differential equations, in FORTRAN (Y. F. Chang), Apr 215

AT&T Information Systems
Model 4000 modem from, Dec 255
Model 4024 modem from, Dec 255
PC 6300 computer from, Hardcard hard disk for, May 273
UNIX PC from, review (A. J. W. Mayer), May 254

Audio-and-video multiplexer, construction of (S. Ciarcia), Feb 85

Automata, cellular, Dec 181
origins of, Dec 182

Automation of chemistry laboratory (G. W. Kramer, P. L. Fuchs), Jan 263

AUTO.OPS implementation of OPS5 programming language, Nov 222

AVMUX audio-and-video multiplexer, construction of (S. Ciarcia), Feb 85

Aztec C
Badfile utility in (L. Baker), Feb 157
Manx Aztec C68K C compiler for Amiga, review (C. Heath), Nov 271

B

Backup systems, tape comparison of (A. Antonuccio), May 227
hardware reviews (M. C. Rubel), Oct 243

Badfile utility in C language for CP/M systems (L. Baker), Feb 157

Balance of Power game (E. Shapiro), Mar 299, Dec 322

Bank-switched memory-expansion systems
AST Enhanced Expanded Memory Specification, IBM 170
Lotus/Intel/Microsoft Expanded Memory Specification (R. Duncan), IBM 169

BASIC
abstract mathematical art in (K. E. Perry), Dec 181
best of BIX on, IBM 304
bit-mapped classifier system on house architecture in, Nov 168
calculating impact of nuclear weapons with programs in (J. R. Fanchi), Dec 143
Circuit Cellar BASIC computer/controller-30 analog-to-digital converter board (S. Ciarcia), Jan 105
Circuit Cellar BASIC computer/controller-52, construction of analog-to-digital converter compatible with (S. Ciarcia), Jan 105
COLOR3D.BAS program displaying molecules in color in, for IBM PC (J. J. Farrell), Feb 149
Diophantine equation solver program in, Mar 346
Euclid's algorithm in, Jan 399
freeform curve program in, for Texas Instruments Professional Computer (S. Enns), Dec 225
Getspec assembly language routine for passing filenames to (B. Hubanks), Nov 119
Hilbert curve program in (M. Ackerman), June 137
keyed file access in (S. C. Perry), Sept 137
Macintosh Explorer disassembler program in (O. Andrade), Mar 145
Mapper similarity mapping program for Macintosh in (R. Spencer), Aug 85
number-sequence games in (R. T. Kurosaka), Aug 333
Pan and Reif algorithm for inversion of large matrices in, Apr 184
physical property estimation programs in (J. N. Stone), July 253
probability analysis programs in, on sucker bets (R. T. Kurosaka), Nov 373
ProDOS catalog sort routine in Applesoft BASIC, June 117
Runge-Kutta method for solving differential equations in (D. M. Leo), Apr 196
SmartWatch real-time clock program in, Mar 120
stress analysis program for underground mining in, July 222
subroutine overlays in GW-BASIC (M. Carmichael), May 151
text analysis programs identifying unknown authors in (J. Tankard), Feb 231
three-dimensional graphics program in (H. Mittelbach), Jan 153
for molecules in color (J. J. Farrell), Feb 149
on quadric surfaces (G. Haroney), Dec 215
translated into C with CGEN program (D. Pountain), Oct 311
Truss2 bridge-truss analysis program in (C. Pedicini), July 145
ZBasic interactive BASIC compiler, review (T. J. Byers), May 265

Batteries Included DEGAS painting software, Apr 320

B&C Microsystems Model 1409 EPROM programmer, review (R. Jacobs), May 279

Benchmarks
on Amiga computer model 1000, May 254, Oct 234
on Apricot XEN computer, Apr 309
on Arity/Prolog version 3.2, Mar 248
on Atari 520ST computer, May 354, June 236
on Atari 1040ST computer, Mar 87, May 354
on AT&T UNIX PC computer, May 257
on Canon A-200 computer system, Jan 296
on C. Itoh TriPrinter dot-matrix printer, Sept 284
characteristics and interpretation of (B. Webster), Jan 371
on Commodore 128 personal computer, July 271
on Compaq Deskpro 286 computer, June 246
on Compaq Deskpro 386 computer, Nov 86
on Compaq Portable II Model 3 computer, Oct 240
on Conquest Turbo PC computer, July 290
on Datran Modem Accelerator version 1.1 for text transmission, Aug 292
on Definicon 68020 coprocessor, July 140
on DeSmet C Development Package for Macintosh, Aug 253
on Eco-C88 C compiler, Jan 310
on Epson Equity I computer, Nov 241
on Epson Equity III computer, Dec 241
on filePro 16 database management software, Nov 298
on Franklin ACE 2200 computer, Sept 264
on Fujitsu DL2400 24-pin dot-matrix printer, Nov 257
on GE 3-8100 printer, May 294
on General Computer HyperDrive 2000 computer, Nov 324
on Hardcard hard disk, May 276
on Hercules Graphics Card Plus, Dec 251
on Hochstrasser Modula-2 System for Z80 CP/M systems, Mar 226
on IBM PC accelerators, IBM 157
on IBM PC AT at various CPU speeds (E. White), IBM 212
on IBM Wheelprinter E, July 316
on ITC Modula-2 Software Development System, Oct 256
on ITT XTRA XP computer, July 283
on KAMAS outline processor, Apr 244
on Kaypro 286i computer, Mar 220
on Kaypro PC computer, Nov 241
on Lattice C compilers for Amiga, Nov 271
for IBM PC, Feb 277
on Leading Edge Model D PC computer, Sept 270
on Levco Prodigy 4 computer, Nov 324
on LISP implementations BYSO LISP and Waltz LISP, July 294
on MacCharlie accessory for Macintosh, Feb 266
on Macintosh 512K version, Nov 324
compared to Atari and

Amiga computers, May 354
and mass storage devices, June 354
Plus version, May 354, Nov 248, Nov 324
on Manx Aztec C68K C compiler for Amiga, Nov 271
on Microsoft Word version 3.0 word processor, Oct 262
on Mix C Compiler, June 258
on modems, Dec 255
on Motorola VME/10 system, Feb 256
on Multitech MPF-PC/700 D1 computer, Nov 241
on NCR PC6 computer, Aug 242
on NewWord 3 word processor, Aug 274
on Nisso NP-2410 24-pin dot-matrix printer, Nov 257
on Panasonic Exec. Partner transportable computer, Apr 234
on Paradox 1.1 relational database program, Sept 306
on Pascal packages for IBM PC, Dec 274
on PCTEX and MicroTEX typesetting packages, Apr 270
on Pocket APL language, Mar 238
on Scottsdale Systems Color Fox computer system, Jan 304
on Sider hard disk drive, Jan 322
on Sperry PC/IT computer, Aug 249
on tape backup units, Oct 247
on TeleVideo Tele-286 Model 2 computer, June 254
on TOPSI 2.0 programming language for IBM PC, Aug 261
on Toshiba P321 24-pin dot-matrix printer, Nov 257
on Turbo Pascal version 3.0, Feb 282
on Turbo Prolog programming language, Sept 295
on Western AT computer, Dec 241
on WordPerfect 4.1 word processor, Sept 312
on Xerox 6060 computer, Sept 277
on ZBasic compiler, May 267
on Zenith Z-241 computer, Dec 241
on Zenith Z-248 computer, Dec 241

Bernoulli Box for Macintosh (B. Webster), Feb 344

Bézier curves, with BASIC program for Texas Instruments Professional Computer (S. Enns), Dec 225

Binary numbers, algorithm for extracting *n*th root from (L. S. Wo), Nov 115

BIOS, Extended (XBIOS) of Atari ST computers, Sept 223

Bit-mapped classifier (P. W. Frey), Nov 161
on house architecture, in BASIC, Nov 166
on political predictions (P. A. Schrodt), Nov 177

Bit Pad Two digitizer, review (E. D. Hearn), Nov 261

BIZCOMP IntelliModem EXT, Dec 255

Blind users
Braille-Edit talking word processor for, Mar 199
review (H. Brugsch), Mar 251
workable computing systems for (A. Arditi, A. E. Gillman), Mar 199

Blue Chip financial investment simulation software (E. Shapiro), Dec 326

Book reviews
Abelson, H., Sussman, G. J., and Sussman, J. *Structure and Interpretation of Computer Programs*, Nov 70
Alber, A. F. *Videotex/Teletext: Principles and Practices*, Feb 57
Ammeraal, L. *Programming Principles in Computer Graphics*, Dec 68
Andriole, S. J., ed. *Software Validation, Verification, Testing and Documentation (A Source Book)*, Nov 76
Bentley, J. *Programming Pearls*, Oct 68
Berry, P. *Operating the IBM PC Networks*, IBM 26
Busch, D. D. *Program Your IBM PC to Program Itself!*, IBM 32
Byers, T. J. *Inside the IBM PC AT*, IBM 26
Campbell, J. *Crafting C Tools for the IBM PCs*, IBM 34
Chamberlin, H. *Musical Applications of Microprocessors*, June 63
Chernicoff, S. *Macintosh Revealed, Volume One: Unlocking the Toolbox*, Sept 74
Chernicoff, S. *Macintosh Revealed, Volume Two: Programming with the Toolbox*, Apr 67, Sept 74
Commander, J. *Macintosh Assembly Language Programming*, Sept 74
Critchlow, A. J. *Introduction to Robotics*, Jan 57
DeVoney, C. *Using PC-DOS*, IBM 30
Dodge, C., and Jerse, T. A. *Computer Music: Synthesis, Composition and Performance*, June 67
Donnelly, D. P. *The Computer Culture*, Mar 58
Dreyfus, H. L., and Drefus, S. E. *Mind Over Machine*, Aug 70
Duncan, R. *Advanced MS-DOS*, IBM 23
Eggebrecht, L. C. *Interfacing to the IBM Personal Computer*, IBM 26
Fortgang, K. S., ed. *M68000 8-/16-/32-Bit Microprocessors Programmer's Reference Manual*, 5th ed., Sept 70
Foster, D. L. *The Practical Guide to the IBM Personal Computer AT*, IBM 28
Gilburne, M. R., Johnston, R. L., and Grogan, A. R., eds. *The Computer Law Annual 1985*, Mar 74
Gust, P. J. *Introduction to Machine and Assembly Language Programming*, July 70
Kamin, J. *MS-DOS Power User's Guide*, IBM 30
Kelleher, K., and Cross, T. B. *Teleconferencing: Linking People Together*, Feb 60
Kelly-Bootle, S., and Fowler, B. *68000, 68010, 68020 Primer*, Sept 70
King, R. A. *The MS-DOS Handbook*, IBM 30
Koçak, H. *Differential and Difference Equations Through Computer Experiments*, July 63
Lambert, S., and Ropiequet, S., eds. *CD ROM: The New Papyrus*, Oct 65
Manning, P. *Electronic and Computer Music*, June 76
McFadden, F. R., and Hoffer, J. A. *Data Base Management*, Jan 68
Merry, M., ed. *Expert Systems 85: Proceedings of the Fifth Technical Conference of the British Computer Society Specialist Group on Expert Systems*, Nov 65
Miller, A. R. *Assembly Language Techniques for the IBM PC*, IBM 34
Mims, F. M. III. *Siliconnections: Coming of Age in the Electronic Era*, Feb 74
Mize, R. R., ed. *The Microcomputer in Cell and Neurobiology Research*, Jan 60
Morse, S. P., and Alpert, D. J. *The 80286 Architecture*, IBM 28
Mortimore, E. F. *Amiga Programmer's Handbook*, Sept 65
Norton, P. *The Peter Norton Programmer's Guide to the IBM PC*, Mar 57
Penna, M. A., and Patterson, R. R. *Projective Geometry and Its Applications to Computer Graphics*, Dec 65
Ramirez, R. W. *The FFT: Fundamentals and Concepts*, Apr 57
Roads, C., and Strawun, J., eds. *Foundations of Computer Music*, June 70

Rochkind, M. J. *Advanced UNIX Programming,* Oct 74
Rosenzweig, E., and Harrison, H. *Programming the 68000: Macintosh Assembly Language,* Sept 76
Scanlon, L. J. *80286 Assembly Language on MS-DOS Computers,* IBM 36
Schnapp, R. L. *Macintosh Graphics in Modula-2,* May 68
Schustack, S. *Variations in C,* May 63
Schwaderer, W. D. *The C Wizard's Programming Reference,* July 66
Shafer, D. *Pascal Primer for the Macintosh,* Feb 66
Siechert, C., and Wood, C. *The Power of Running PC/DOS,* IBM 32
Sisk, J. E., and VanArsdale, S. *Exploring the Pick Operating System,* Apr 70
Stevens, A. *C Development Tools for the IBM PC,* IBM 36
Stroustrup, B. *The C++ Programming Language,* Aug 63
Szczepanowski, N., and Gunther, B. *Atari ST GEM Programmer's Reference,* Sept 78
Takatsuka, J., Burnard, D., and Huxham, F. *Using the Macintosh Toolbox with C,* Sept 76
Thomas, R., Rogers, L. R., and Yates, J. L. *Advanced Programmer's Guide to UNIX System V,* Aug 68
Triebel, W. A., and Singh, A. *The 68000 Microprocessor: Architecture, Software, and Interfacing Techniques,* Sept 74
Troy, D. A. *Complete C Language Programming for the IBM PC,* IBM 38
Waite, M., Lafore, R., and Lansing, I. *Microsoft Macinations,* Mar 62
Ward, T. A. *Programming C on the Macintosh,* Sept 76
Welsh, J., and Hay, A. *A Model Implementation of Standard Pascal,* Dec 70
William, C., and Kane, J. *68000 Microprocessor Handbook: Includes 68008, 68010, & 68020,* 2nd ed., Sept 68
Williams, C. S. *Designing Digital Filters,* Apr 60
Williams, S. *Programming the Macintosh in Assembly Language,* Sept 76
Wirth, N. *Algorithms and Data Structures,* May 74
Wolverton, V. *Running MS-DOS,* IBM 32

Borland International
Reflex analytical database program from, review (R. DeMaria), Aug 277
Turbo Editor Toolbox from (E. Shapiro), Mar 297
Turbo Lightning version 1.00A spelling checker from, review (R. Ramsey), Nov 289
Turbo Pascal from for Macintosh (B. Webster), Nov 338
version 3.0, review (M. Bridger), Feb 281
Turbo Prolog programming language from (B. Webster), Sept 335
review (N. C. Shammas), Sept 293

Boxcalc spreadsheet program (E. Shapiro), May 336

Boxes and Arrows spreadsheet and graphics program (E. Shapiro), May 335

Braille-Edit talking word processor, Mar 199
review (H. Brugsch), Mar 251

Breakout game, three-dimensional version for Amiga (B. Webster), Dec 310

BREAKPT routine for PC-DOS DEBUG program, programming of (E. Batutis), Sept 127

Bridge construction, computer-aided engineering in. *See* Engineering, computer-aided

Britain, BYTE reports from. *See* U.K., BYTE reports from

Brown & Co. PC-Pedal input device, review (C. H. Pappas), May 285

B-spline curves, with BASIC program for Texas Instruments Professional Computer (S. Enns), Dec 225

Bulletin board system in Japanese language (W. M. Raike), Nov 348

Businessoft Mindreader word processor (E. Shapiro), Aug 319

Business systems
Abundance FORTH-based database language for, in public domain (R. Green), Oct 193
Intuitive solution business application generator for (D. Pountain), May 363

BYSO LISP version 1.17, review (W. G. Wong), July 293

BYTE listings
on abstract mathematical art, Dec 186
on anagram solving in Pascal, July 113
on assembly routines in MS-FORTRAN programs, IBM 222
on ATOMCC toolbox for solving differential equations, in FORTRAN, Apr 215
on AUTO.OPS implementation of OPS5 programming language, Nov 222
on Badfile utility in C language for CP/M systems, Feb 162
on bit-mapped classifier system
on house architecture, in BASIC, Nov 168
on political predictions, in Turbo Pascal, Nov 188
on BREAKPT routine for PC-DOS DEBUG program, Sept 127
on calculating impact of nuclear detonations, in BASIC, Dec 152
on Circuit Cellar intelligent serial EPROM programmer, Oct 119
on COLOR3D.BAS program for displaying molecules in color, Feb 152
on CP/M public domain software, Oct 222
on cyclic redundancy check calculations in C, Sept 120
on debugging program for 8031 in-circuit emulator, July 190
on Decision Support System in Prolog, Nov 201
on DRAGON program in FORTH, Apr 138
on DVORAK.BAS program for studying finger motion in typing, Feb 244
on EDITNET program in LISP, Feb 196
on EditSort capsule editor and QuickSort capsule, Jan 148
on Enhanced Console Driver for user-friendly interface to DOS, Oct 183
on Euclid's algorithm, Jan 399
on Expanded Memory Specification EMSDISK driver, IBM 174
on file-indexing program in Turbo Pascal, June 102
on frame-manipulation routines in Prolog, Jan 238
on freeform curves, with BASIC program for Texas Instruments Professional Computer, Dec 230
on graphing quadric surfaces, with BASIC program for IBM PC, Dec 224
on Henon mapping in Turbo Pascal, Dec 163
on Hilbert curve program in BASIC, June 138
on Huffman coding for data compression, in Modula-2, May 108
on Icon high-level programming language, Oct 167
on Induce program for extracting knowledge from data, in Turbo Pascal, Nov 158
on integral approximation with Aitken extrapolation, Dec 113
on interrupt routines for IBM PC in assembly language, IBM 252
on keyed file access in BASIC, Sept 143
on Macintosh Explorer disassembler program in BASIC, Mar 148
on MacView program for transferring Macintosh graphics to IBM PC, in

Pascal, June 131
on Mandelbrot program for Amiga, in Lattice C, Dec 208
on Mapper similarity mapping program in BASIC, Aug 92
on Marvin program for teaching computers to learn, in Prolog, Nov 225
on material selection program for construction, in Turbo Pascal, July 235
on memory manipulations for IBM PC, IBM 236
on Microsafe finite-element-analysis programs, in FORTRAN, July 209
on MIDI programs for IBM PC
in C, June 204
in Turbo Pascal, June 224
on musical fractals, in BASIC, June 185
on operating system functions from Turbo Pascal, Dec 103
on Pan and Reif algorithm for inversion of large matrices, in BASIC, Apr 184
on PD PROLOG, Oct 165
on Performance Index benchmark, IBM 158
on performance programming for IBM PC-compatible products, IBM 194
on physical property estimation programs in BASIC, July 253
on poetry processing program in Pascal, Feb 225
on polar method for generating normal deviates, in Pascal, Aug 131
on ProDOS catalog sort routine in BASIC, June 122
on protected-mode program for IBM PC AT, IBM 123
on RAM-loadable character sets for IBM PC, IBM 198
on Runge-Kutta method for solving differential equations, in FORTRAN, Apr 196
on shareware, Oct 297
on SIMPL compiler, Jan 144, Feb 105
on SmarTime code in BASIC, Mar 124
on SPICE program for Commodore 64, July 178
on spreadsheet program in Modula-2, for Macintosh, July 97
on storage allocators, in Modula-2, Oct 123
on stress analysis program for underground mining, in BASIC, July 222
on subroutine overlays in GW-BASIC, May 153
on text analysis programs for identification of unknown authors, Feb 238
on three-dimensional graphics programs, in BASIC, Jan 153, Feb 152
on Truss2 bridge-truss analysis program in BASIC, July 145
on Turbo Pascal programs version 3.0, Feb 286
on Turbo Prolog benchmarks, Sept 295
on Visual Syntax editor in LISP, Feb 144
on Watson deductive reasoning program in Prolog, Nov 214
on write-once database, May 199
on Z80MU program for emulation of Zilog Z80 and CP/M 2.2, Oct 203

C

C language
ANSI standards on (S. A. Hersee, D. Knopoff), Mar 135
Badfile utility in (L. Baker), Feb 157
BASIC translated into, with CGEN program (D. Pountain), Oct 311
compared to assembly language (T. Hogan), IBM 267
compatibility with UNIX and MC68000 architecture, Sept 192
cyclic redundancy check calculations in, Sept 120
DeSmet C development package (W. M. Raike), June 339
review (J. Robie), Aug 253
Easy C preprocessor (P. Orlin, J. Heath), May 137
Eco-C88 C compiler, review (D. D. Clark), Jan 307
IBM PC-compatible programs in (J. Rosenblum, D. Jacobs), IBM 181
Lattice C compiler for Amiga, review (C. Heath), Nov 271
for IBM PC, version 2.15, review (D. S. Woolston), Feb 273
Let's C and csd C development package and symbolic debugger, review (W. G. Wong), Aug 267
Lightspeed C development environment for Macintosh (B. Webster), Aug 323, Sept 340
Mac C compiler for Macintosh, Apr 318
Mandelbrot program for Amiga in (P. B. Schroeder), Dec 207
Manx Aztec C68K C compiler for Amiga, review (C. Heath), Nov 271
Megamax C development system for Atari ST (B. Webster), Nov 326
MIDI programs for IBM PC in, June 204
Mix C compiler for IBM PC, review (R. Grehan), June 257

C++ object-oriented programming language, Aug 196

Camera systems in machine vision (P. Dunbar), Jan 161

Canon A-200 computer system, review (P. V. Callamaras), Jan 293

Capsule editors, for customizing modules in Modula-2 (N. C. Shammas), Jan 145

Catalog sort routine, in Applesoft BASIC (A. C. Silvestri), June 117

CD-ROM
compared to magnetic disks (B. Zoellick), May 177
databases available on (N. Desmarais), May 235
development of, May 163
logical format standards on, May 184
new products related to, May 218
software development for (B. Zoellick), May 177
technology of, May 164

Cellular automata, Dec 181
origins of, Dec 182

Central Point Software PC Tools version 1.10, review (R. Rabinovitz), Oct 265

Cermetek Security Modem, Dec 255

CGEN program for translation of BASIC into C (D. Pountain), Oct 311

Character sets for IBM PC, RAM-loadable (R. Wilton), IBM 197

Chebyshev approximations, Apr 174

Chemistry laboratory, automation of (G. W. Kramer, P. L. Fuchs), Jan 263

CHEMMOD molecular modeling system (D. Pountain), Dec 334

Ciarcia's Circuit Cellar (S. Ciarcia)
on addition of SCSI to SB180 computer, May 85, June 107
on analog-to-digital converter, Jan 105
on AVMUX audio-and-video multiplexer, Feb 85
on data encryptor, Sept 97
on GT180 color graphics board
basic technology of, Nov 105
hardware of, Dec 87
on intelligent serial EPROM programmer, Oct 103
on overzealous computer security, Apr 85
on parallel interfacing applications of, Aug 97
introductions to, July 85
on real-time clocks, Mar 113

CIE Terminals C. Itoh Triprinter Model 20, review (R. D. Swearengin), Sept 283

C. Itoh TriPrinter dot-matrix printer, review (R. D. Swearengin), Sept 283

Circuit Cellar projects. *See* Ciarcia's Circuit Cellar

Circuit simulation
programs
for Commodore 64 (D. McNeill), July 170
introduction to (W. Blume), July 165
SPICE (Simulation Program for Integrated Circuit Engineering), July 165, July 170

Clocks, real-time (S. Ciarcia), Mar 113

CLUEDO, Watson deductive reasoning program in Prolog based on (J.-C. Emond, A. Paulissen), Nov 207

Coal mining, stress analysis of tunnels in (D. L. Petersen, S. L. Crouch), July 219

Codex Corp Model 2233 modem, Dec 255

College education at home, with Electronic University Network (D. Osgood), Mar 171

Color Fox computer system from Scottsdale Systems, review (J. D. Unger), Jan 301

COLOR3D.BAS program for displaying molecules in color, for IBM PC (J. J. Farrell), Feb 149

COMDEX show of 1986 in Japan (W. M. Raike), July 371

ComicWorks painting program for Macintosh (E. Shapiro), Dec 321

COMM180 1200-bps modem and SCSI expansion board for SB180 computer (S. Ciarcia), May 85, June 107

Commodore computers
Amiga. *See* Amiga computer from Commodore
Commodore 64
MIDI interfaces for, review (R. Powell, R. Grehan), June 265
SPICE circuit analysis program for (D. McNeill), July 170
Commodore 128
personal computer, review (W. Wiese Jr.), July 269

Communications
comparison of modems in (S. Satchell), Dec 255
with Datran Modem Accelerator version 1.1 for text transmission, review (B. Nance), Aug 289
with Enable integrated software package, Jan 331
with Flash-Com electronic mail and telecommunications software system, review (B. N. Meeks), Dec 281
with Hayes Smartmodem 2400 (W. M. Raike), June 339
with Miracle Technology WS3000 modem (D. Pountain), June 319
with Mirror telecommunications program, Apr 329
with Pibterm shareware program, Oct 300
with ProComm shareware program, Oct 300
Prolog strategy for (L. Su), Oct 160
with Qmodem shareware program, Oct 300

Compact-disk read-only memory. *See* CD-ROM

Compaq computers
Compaq Deskpro 286 Models 1 and 2, review (S. Miastkowski), June 243
Compaq Deskpro 386, preview (T. Thompson, D. Allen), Nov 84
Compaq Portable II Model 3, review (S. Miastkowski), Oct 239
Hardcard hard disk for, May 273

Compilers
in DeSmet C development package (W. M. Raike), June 339
review (J. Robie), Aug 253
Eco-C88 C compiler, review (D. D. Clark), Jan 307
in ITC Modula-2 Software Development System, review (M. Bridger), Oct 255
Lattice C compilers
for Amiga, review (C. Heath), Nov 271
for IBM PC version 2.15, review (D. S. Woolston), Feb 273
in Let's C and csd development package and symbolic debugger, review (W. G. Wong), Aug 267
Lightspeed C development environment for Macintosh (B. Webster), Aug 323, Sept 340
Mac C compiler for Macintosh, Apr 318
MacLanguage Series Pascal compiler (B. Webster), Feb 340, Apr 315
Manx Aztec C68K C compiler for Amiga, review (C. Heath), Nov 271
Mix C compiler for IBM PC, review (R. Grehan), June 257
Modula-2 System for Z80 CP/M systems, review (B. R. Anderson), Mar 225
PIE (Prolog Inference Engine) editor and compiler (S. Y. Blackwell), Oct 164
SIMPL
extensions of (J. Amsterdam), Feb 103
procedures and functions of (J. Amsterdam), Jan 131
TDI Modula-2/ST compiler (B. Webster), Feb 332
Turbo Prolog (N. C. Shammas), Sept 293
ZBasic interactive BASIC compiler, review (T. J. Byers), May 265

Computex show of 1986 in Taiwan (W. M. Raike), Oct 307

ConcertWare+ music software for Macintosh, review (M. S. Bernardo), June 273

Concurrency, in multiprocessing operating systems, May 114

Conquest Turbo PC computer, review (J. D. Unger), July 289

Console Utility command syntax, Oct 185

Consulair Mac C compiler version 4.01, Apr 318

Construction, computer-aided engineering in. *See* Engineering, computer-aided

Convertible IBM PC, description of (G. M. Vose), IBM 83

Coprocessors
coprocessor accelerator boards for IBM PC, IBM 144
Definicon 68020 coprocessor (T. Marshall, C. Jones, S. Kluger), July 120, Aug 108

Coral Software Object Logo object-oriented language for Macintosh, Aug 184

Cotton Software Boxcalc program (E. Shapiro), May 336

CP/M operating systems
Badfile utility for (L. Baker), Feb 157
best of BIX on, Oct 338
public domain software for (B. N. Meeks), Oct 219
Z80 based. *See* Z80-based CP/M systems

CRC (cyclic redundancy check) calculation, programming project (G. Morse), Sept 115

Cruise Control cursor control program (E. Shapiro), Nov 361

Cryptography
background information on, Sept 98
Circuit Cellar data encryptor (S. Ciarcia), Sept 97
glossary on, Sept 111

Crystal change enhancing IBM PC AT performance, IBM 209

CTS-Fabri-Tek modems, Dec 255

Cursor control with Cruise Control program (E. Shapiro), Nov 361

Curves
free-form, with BASIC program for Texas Instruments Professional Computer (S. Enns), Dec 225
Hilbert curve program in BASIC, June 138

CWare DeSmet C development package (W. M Raike), June 339
review (J. Robie), Aug 253

Cyclic redundancy check (CRC) calculation, programming project (G. Morse), Sept 115

D

Dac-Easy Word word processor (E. Shapiro), July 381

Daisy-wheel printer, IBM Wheelprinter E, review (R. D. Swearengin), July 315

Dallas Semiconductor SmartWatch real-time clock, Mar 118

Dark Castle game for Macintosh (E. Shapiro), Dec 327

DASCH external RAM disk for Macintosh (B. Webster), June 353

Data
- compression with Huffman coding (J. Amsterdam), May 99
- design principles in organization of files (L. Shapiro), Apr 129
- encryption
  - background information on, Sept 98
  - with Circuit Cellar data encryptor (S. Ciarcia), Sept 97
  - glossary on, Sept 111
- entry, digitizers for, comparison of (E. D. Hearn), Nov 261
- flow in functional programming (C. Hankin, D. Till, H. Glaser), May 123
- handling conflicts in, with decision support system in Prolog (C. Y. Cuadrado, J. L. Cuadrado), Nov 193
- ID3 algorithm for extracting knowledge from (B. Thompson, W. Thompson), Nov 149
- protection of, with tape backup systems, May 227, Oct 243
- recovery and restoration of, with Norton Utilities, PC Tools and Super Utility programs, comparison of (R. Rabinovitz), Oct 265
- structures in object-oriented programming, Aug 142

Database
- Abundance Forth-based database language, in public domain (R. Green), Oct 193
- in coin-operated terminals in Japan (W. M. Raike), Nov 347
- data design principles in organization of (L. Shapiro), Apr 129
- Enable integrated software package for management of, Jan 331
- filePro 16 and filePro 16 Plus database management software, review (R. Harkness), Nov 297
- Instant Recall Database manager (E. Shapiro), Oct 297
- on optical disks (N. Desmarais), May 235
- Paradox 1.1 relational database program, review (R. DeMaria), Sept 303
- Reflex analytical database program, review (R. DeMaria), Aug 277

Data Show of 1985 in Japan, highlights of (W. M. Raike), Feb 317

Datatek Datatext word processor (E. Shapiro), July 383

Data Translation
- DT2801 board from, software interfacing packages for (P. Wirth, L. E. Ford), July 303
- PCLAB software from, July 303

Datran Modem Accelerator version 1.1, review (B. Nance), Aug 289

Dayna Communications MacCharlie Macintosh accessory, review (L. Crockett), Feb 262

dBASE III Plus software (E. Shapiro), June 330

Debugging
- with 8031 in-circuit emulator (G. Dinwiddie), July 181
- with Advanced Trace86 assembler/debugger (B. Webster), Oct 293
- with Let's C and csd C development package and symbolic debugger, review (W. G. Wong), Aug 267
- of Macintosh applications (J. West), Dec 127
- with Metascope program for Amiga (B. Webster), Nov 334
- of optical disks, May 198
- codes used in (S. W. Golomb), May 203
- with PC-DOS DEBUG program and BREAKPT routine (E. Batutis), Sept 127
- with Professional Debug Facility and Advanced Fullscreen Debug programs, comparison of (J. C. Carden), Apr 249

Decimals, repeating, converted to fractions (R. T. Kurosaka), Jan 397

Decision support system in Prolog (C. Y. Cuadrado, J. L. Cuadrado), Nov 193

Deductive reasoning with Watson program in Prolog (J.-C. Emond, A. Paulissen), Nov 207

Definicon 68020
- coprocessor hardware and operating system of (T. Marshall, C. Jones, S. Kluger), July 120
- kernel functions of, July 142
- software for (T. Marshall, C. Jones, S. Kluger), Aug 108

DEGAS painting software for Atari 520ST (B. Webster), Apr 320

Deluxe Paint program for Amiga, Apr 322

Deskpro computers from Compaq
- model 286, review (S. Miastkowski), June 243
- model 386, preview (T. Thompson, D. Allen), Nov 84

DeSmet C development package (W. M. Raike), June 339
- for Macintosh, review (J. Robie), Aug 253

DevpacST 68000 assembly language development package (B. Webster), Nov 330

Differential equations
- ATOMCC toolbox method for solving, in FORTRAN (Y. F. Chang), Apr 215
- Runge-Kutta method for solving, Apr 216
  - in BASIC (D. M. Leo), Apr 196
  - in FORTRAN (B. Thomas), Apr 191

Digi-Pad 5 digitizer for data entry, review (E. D. Hearn), Nov 261

Digital music synthesis (R. A. Moog), June 155
- with Kurzweil 250 Digital Synthesizer, review (C. Morgan), June 279
- with Macintosh (C. Yavelow), June 171

Digital Research GEM (Graphics Environment Manager)
- in Atari ST computers (M. Rothman), Sept 223
- and GEM Draw painting program, review (R. Birmele), May 269

Digital typography with TEX typesetting system, Feb 206
- Knuth interview on (G. M. Vose, G. Williams), Feb 169

Digitizers for data entry, comparison of (E. D. Hearn), Nov 261

Diophantine equations (R. T. Kurosaka), Mar 343

DirectoryDemo program, Dec 103

Disabilities
- in hearing impairment, Kurzweil Voice Writer in (R. Kurzweil), Mar 177
- in motion impairment, robotic aids in (K. G. Engelhardt, R. Edwards), Mar 191
- in speech impairment, semantic compaction system in (B. R. Baker), Mar 160
- in visual impairment
  - assembling workable computing system in (A. Arditi, A. E. Gillman), Mar 199
  - Braille-Edit talking word processor in, Mar 199, 251

Disassembler program for Macintosh, Macintosh Explorer, programming of (O. Andrade), Mar 145

Disks
- compact-disk read-only memory. *See* CD-ROM
- hard. *See* Hard disks
- magnetic, compared to compact-disk read-only memories (B. Zoellick), May 177
- optical. *See* Optical disks

Displays
- gas-plasma, of Panasonic Exec. Partner transportable computer, Apr 231

size and shape of characters in, with RAM-loadable character sets for IBM PC (R. Wilton), IBM 197

Dot-matrix printers
C. Itoh Triprinter, review (R. D. Swearengin), Sept 283
Fujitsu DL2400 24-pin, review (R. D. Swearengin), Nov 255
GE 3-8100, review (R. D. Swearengin), May 293
Lettrix resident print processor used with, review (A. R. Miller), May 299
Nissho NP-2410 24-pin, review (R. D. Swearengin), Nov 255
Toshiba P321 24-pin, review (R. D. Swearengin), Nov 255

DRAGON program for Macintosh, in Forth (B. R. Land), Apr 137

DS-AP MIDI interface for Apple II computers, review (R. Powell, R. Grehan), June 265

DSC-64 MIDI interface for Commodore 64, review (R. Powell, R. Grehan), June 265

DSI-020 coprocessor board from Definicon Systems
hardware and operating system of (T. Marshall, C. Jones, S. Kluger), July 120
software for (T. Marshall, C. Jones, S. Kluger), Aug 108

Dvorak keyboard layout, efficiency of (D. W. Olson, L. E. Jasinski), Feb 241

DynaMac Japanese version of Macintosh, Jan 382

Dynamic load balancing in parallel computing systems (D. Pountain), July 368, Sept 359

Dynamic Master Systems TOPSI 2.0 programming language for IBM PC, review (L. Moskowitz), Aug 261

E

Easy C preprocessor (P. Orlin, J. Heath), May 137

Easy word processor from MicroPro (E. Shapiro), Mar 300

Echo speech synthesizers from Street Electronics, Mar 199, Mar 251, Mar 261

Ecosoft Eco-C88 C compiler
review (D. D. Clark), Jan 307
update on, Jan 314

EDITNET program (J. Pollack), Feb 196

EditSort capsule editor, Jan 148

Education at home, with Electronic University Network (D. Osgood), Mar 171

Elderly, robotic aids increasing independence of (K. G. Engelhardt, R. Edwards), Mar 191

Electronic Arts software for Amiga (B. Webster), Apr 322
Instant Music program (B. Webster), Dec 308

Electronic mail, with Flash-Com electronic mail and telecommunications software system, review (B. N. Meeks), Dec 281

Electronic University Network (D. Osgood), Mar 171

EM/3+ software from Megasoft, May 329

Emulation
with emulator accelerators for IBM PC, IBM 150
of EPROM with Emulo-8 (S. R. Ball), Apr 105
of Intel 8031 processor, construction of emulator board for (G. Dinwiddie), July 181
of Zilog Z80 and CP/M 2.2 with Z80MU program (R. A. Baumann), Oct 203

Emulo-8 EPROM emulator, construction of (S. R. Ball), Apr 105

Enable integrated software package
version 1.0, review (S. King), Jan 331
version 1.1 (R. Malloy), Jan 334

Encryption of data
background information on, Sept 98
with Circuit Cellar data encryptor (S. Ciarcia), Sept 97
glossary on, Sept 111

Engineering, computer-aided
with 8031 in-circuit emulator (G. Dinwiddie), July 181
with circuit simulation programs
for Commodore 64 (D. McNeill), July 170
introduction to (W. Blume), July 165
SPICE (Simulation Program for Integrated Circuit Engineering), July 165, 170
finite-element analysis method in, July 145, 148
on IBM PC (R. W. Johnson, F. G. Loygorri), July 199
in Truss2 bridge-truss analysis program, July 145
as issue theme (G. M. Vose), July 163
with material selection program in Turbo Pascal (T. Sawyer, M. Pecht), July 235
with physical property estimation programs (J. N. Stone), July 253
with stress analysis program for underground mining (D. L. Petersen, S. L. Crouch), July 219
with Truss2 bridge-truss analysis program in BASIC (C. Pedicini), July 145
with U-MAN 1000 computer (D. Pountain), Dec 329

England BYTE reports. *See* U.K. BYTE reports

Enhanced Console Driver, for user-friendly interface to DOS (A. Zackin), Oct 183

EPROM programmers
Circuit Cellar intelligent serial EPROM programmer, construction of (S. Ciarcia), Oct 103
Emulo-8 EPROM emulator, construction of (S. R. Ball), Apr 105
Model 1409 from B&C Microsystems, review (R. Jacobs), May 279

Epson computers
Equity I, review (J. D. Unger), Nov 239
Equity III, review (W. Rash Jr.), Dec 239
Equity I computer, review (J. D. Unger), Nov 239
Equity III computer, review (W. Rash Jr.), Dec 239
Eratosthenes Sieve benchmark. *See* Benchmarks
Error detection and correction. *See* Debugging

Euclid's algorithm (R. T. Kurosaka), Jan 397, Mar 343

Exec. Partner transportable computer from Panasonic, review (R. Malloy), Apr 231

Exhibits and shows
COMDEX show of 1986 in Japan (W. M. Raike), July 371
Computex show of 1986 in Taiwan (W. M. Raike), Oct 307
Data Show of 1985 in Japan, Feb 317
Microcomputer Show of 1986 in Tokyo (W. M. Raike), Sept 351
Software Show of 1985 in Japan, Feb 317
West Coast Computer Faire of 1986 (B. Webster), Aug 323

Expanded Memory Specification
AST Research, IBM 170
Lotus/Intel/Microsoft (R. Duncan), IBM 169

ExperCommonLISP object-oriented language for Macintosh, Aug 180

ExperOPS5 expert-system programming language for Macintosh, review (W. Jacobs), July 297

ExperTelligence
ExperCommonLISP object-oriented language for Macintosh from, Aug 180
ExperOPS5 expert-system programming language for Macintosh from, review (W. Jacobs), July 297

Expert systems
knowledge representation in. *See* Knowledge representation
OPS5 programming language of. *See* OPS5 programming language
Explorer disassembler program for Macintosh, programming of (O. Andrade), Mar 145

F

***Federalist* papers,** text analysis programs identifying author of (J. Tankard), Feb 231
File(s)
with bad sectors or tracks, Badfile utility for identification and location of (L. Baker), Feb 157
compression of
with Datran Modem Accelerator version 1.1, review (B. Nance), Aug 289
with Huffman coding (J. Amsterdam), May 99
data design principles in organization of (L. Shapiro), Apr 129
Getspec assembly language routine for passing filenames to compiled BASIC (B. Hubanks), Nov 119
indexes of records in
in keyed access method in BASIC, Sept 138
programming project on, in Pascal (B. Webster), June 93
keyed access to data in, in BASIC (S. C. Perry), Sept 137
ProDOS catalog sort routine for locating entries in (A. C. Silvestri), June 177
Q&A software package for management of, preview, Jan 120
recovery and restoration of, with Norton Utilities, PC Tools and Super Utility programs, comparison of (R. Rabinovitz), Oct 265
transmission of, with Datran Modem Accelerator version 1.1, review (B. Nance), Aug 289
FilePro 16 and filePro 16 Plus database management software, review (R. Harkness), Nov 297
Finite-element analysis technique, July 145, 148
on IBM PC (R. W. Johnson, F. G. Loygorri), July 199
in Truss2 bridge-truss analysis program, July 145
Finot Group KeepTrack Plus software (E. Shapiro), June 334
First Class Peripherals Sider hard disk drive, review (D. E. Hall), Jan 319
Flag testing using 2^n property, programming insight (R. C. Arp Jr.), Oct 145
Flash-Com electronic mail and telecommunications software system, review (B. N. Meeks), Dec 281
Floating-point operations, approximations in (S. L. Moshier), Apr 161
FM-16*B* computer from Fujitsu, upgrade of (W. M. Raike), Aug 329
FM-16π portable computer from Fujitsu (W. M. Raike), Jan 384, Mar 330
Fonts
for Macintosh (B. Webster), Feb 340
METAFONT typesetting system, Feb 169, 211
for non-Latin scripts, Feb 204
Footmouse input device, review (C. H. Pappas), May 285
Forbin Project Qmodem shareware program (E. Shapiro), Oct 300
Forth
Abundance database language based on, in public domain (R. Green), Oct 193
DRAGON program for Macintosh in (B. R. Land), Apr 137
object-oriented extensions of (D. Pountain), Aug 227
in queue simulation program, Aug 230
FORTRAN
assembly routines in MS-FORTRAN programs (M. Dahmke), IBM 217
ATOMCC toolbox for solving differential equations in (Y. F. Chang), Apr 215
integral approximation with Aitken extrapolation in, Dec 113
Microsafe finite-element analysis programs in, July 209
Runge-Kutta method for solving differential equations in (B. Thomas), Apr 191
Fractals, musical (C. Dodge, C. R. Bahn), June 185
Fractions, conversion of repeating decimals to (R. T. Kurosaka), Jan 397
Framework II software (E. Shapiro), Apr 331
Franklin ACE 2200 computer, review (A. S. Woodhull), Sept 263
Free-form curves, with BASIC program for Texas Instruments Professional Computer (S. Enns), Dec 225
FREQUENCY ANALYZER programs for identifying unknown authors, Feb 232
FT-60 tape backup unit, review (M. C. Rubel), Oct 243
Fujitsu
DL2400 24-pin dot-matrix printer, review (R. D. Swearengin), Nov 255
FM-16*B* computer upgrade (W. M. Raike), Aug 329
FM-16π portable computer (W. M. Raike), Jan 384, Mar 330
FullPaint program for Macintosh (E. Shapiro), Sept 368
Functional programming, data-flow in (C. Hankin, D. Till, H. Glaser), May 123

G

Games
Animal Game in Smalltalk, Aug 153
for Atari ST (B. Webster), Dec 312
Balance of Power game (E. Shapiro), Mar 299, Dec 322
Dark Castle game for Macintosh (E. Shapiro), Dec 327
number-sequence games in BASIC (R. T. Kurosaka), Aug 333
three-dimensional Breakout game for Amiga (B. Webster), Dec 310
Gas-plasma display, of Panasonic Exec. Partner transportable computer, Apr 231
GEM from Digital Research
in Atari ST computers (M. Rothman), Sept 223
and GEM Draw painting program, review (R. Birmele), May 269
General Computer HyperDrive 2000 computer (B. Webster), Nov 323
General Electric GE 3-8100 printer, review (R. D. Swearengin), May 293
Getspec assembly language routine for passing filenames to compiled BASIC (B. Hubanks), Nov 119
Glossaries
on Amiga animation, Sept 246
on data encryption, Sept 111
on image processing technology, Mar 110
on SCSI bus implementation of SB180 computer, June 112
on video, Jan 168
GP-7 and GP-8 digitizers for data entry, review (E. D. Hearn), Nov 261
Graphics
abstract mathematical art, in BASIC (K. E. Perry), Dec 181
with Amiga computer and Aegis Animator and Images animation and painting programs (W. Block), Nov 285

animation software for (E. A. Ditton, R. A. Ditton), Sept 241
and Deluxe Paint program, Apr 322
and Mandelbrot program in Lattice C (P. B. Schroeder), Dec 207
and ROM Kernel, Feb 116
with Atari 520ST and DEGAS painting software, Apr 320
book reviews related to, Dec 65
with Circuit Cellar GT180 color graphics board and SB180 computer
basic technology of (S. Ciarcia), Nov 105
hardware of (S. Ciarcia), Dec 87
with GEM Draw and MacDraw painting programs, comparison of (R. Birmele), May 269
with Henon mapping program in Pascal (G. Hughes), Dec 161
with Hercules Graphics Card Plus, review (R. Malloy), Dec 249
with IBM PC
best of BIX on, Dec 385
and Boxes and Arrows program, May 335
and MacView program (M. Anacker), June 131
and Mandelbrot program in Turbo Pascal (D. Pountain), Sept 359
as issue theme (C. Weston), Dec 159
with Macintosh
and DRAGON program in Forth(B. R. Land), Apr 137
transferred to IBM PC with MacView program (M. Anacker), June 131
with Mandelbrot program for Amiga in Lattice C (P. B. Schroeder), Dec 207
for IBM PC in Turbo Pascal (D. Pountain), Sept 359
quadric surfaces in, with BASIC program (G. Haroney), Dec 215
with Texas Instruments Professional Computer, and BASIC program on free-form curves (S. Enns), Sept 225
with Texas Instruments TMS34010 Graphics System Processor (C. R. Killebrew Jr.), Dec 193
three-dimensional, BASIC program for (H. Mittelbach), Jan 153
for molecules in color (J. J. Farrell), Feb 149
on quadric surfaces (G. Haroney), Dec 215

Graphics Card Plus from Hercules, review (R. Malloy), Dec 249

Graphics System Processor, Texas Instruments TMS34010 (C. R. Killebrew Jr.), Dec 193

Great Wave Software ConcertWare+ music program for Macintosh, review (M. S. Bernardo), June 273

GT180 color graphics board for SB180 computer
basic technology of (S. Ciarcia), Nov 105
hardware of (S. Ciarcia), Dec 87

GTCO Corp digitizer for data entry, review (E. D. Hearn), Nov 261

Guru integrated software package, review (E. R. Tello), Aug 281

GW-BASIC, subroutine overlays in (M. Carmichael), May 151

H

Hammerlab Lettrix resident print processor, review (A. R. Miller), May 299

Hamming code for optical disk error correction (S. W. Golomb), May 203

Hardcard hard disk, review (E. White), May 273

Hard disks
Hardcard, review (E. White), May 273
MacBottom for Macintosh, June 348
MicroBotics MAS-20 hard disk drive for Amiga (B. Webster), Dec 306
Sider for Apple II computers, review (D. E. Hall), Jan 319
tape backup systems for
comparison of (A. Antonuccio), May 227
hardware reviews (M. C. Rubel), Oct 243

Hardware
of 8031 in-circuit emulator (G. Dinwiddie), July 181
in Circuit Cellar projects. *See* Ciarcia's Circuit Cellar
of Definicon 68020 coprocessor (T. Marshall, C. Jones, S. Kluger), July 120
of Emulo-8 EPROM emulator (S. R. Ball), Apr 105
IBM PC accelerator boards (S. S. Fried), IBM 141
increasing speed of IBM PC AT (B. K. Roemmele), IBM 209
of Lotus/Intel/Microsoft Expanded Memory Specification (R. Duncan), IBM 169
Macintosh Plus memory-expansion kits, comparison of (C. Crawford, T. Thompson), Nov 250
memory management units for 68000 architectures (G. Zehr), Nov 127
of micro-based supercomputer (N. H. Christ, A. E. Terrano), Apr 145
in MIDI interface
for IBM PC, construction project (J. Kubicky), June 199
reviews (R. Powell, R. Grehan), June 265
reviews. *See* Hardware reviews
in robotics
in automation of chemistry laboratory (G. W. Kramer, P. L. Fuchs), Jan 263
in tactile sensing (K. E. Pennywitt), Jan 172
in vision system (P. Dunbar), Jan 161
Texas Instruments TMS34010 Graphics System Processor (C. R. Killebrew Jr.), Dec 193

Hardware reviews
on Advantage! multifunction board for IBM PC AT (T. J. Byers), Jan 327
on B&C Microsystems Model 1409 EPROM Programmer (R. Jacobs), May 279
on C. Itoh TriPrinter Model 20 (R. D. Swearengin), Sept 283
on Commodore 128 personal computer (W. Wiese Jr.), July 269
on Datran Modem Accelerator version 1.1 for text transmission (B. Nance), Aug 289
on digitizers for data entry (E. D. Hearn), Nov 261
on Fujitsu DL2400 dot-matrix printer (R. D. Swearengin), Nov 255
on GE 3-8100 printer (R. D. Swearengin), May 293
on Hardcard hard disk (E. White), May 273
on Hercules Graphics Card Plus (R. Malloy), Dec 249
on IBM Wheelprinter E (R. D. Swearengin), July 315
on Kurzweil 250 Digital Synthesizer (C. Morgan), June 279
on MIDI Interfaces (R. Powell, R. Grehan), June 265
on modems (S. Satchell), Dec 255
on Nissho NP-2410 dot-matrix printer (R. D. Swearengin), Nov 255
on non-keyboard input devices (C. H. Pappas), May 285
on Printit printer interface card (H. Brugsch, J. J. Lazzaro), Mar 261
on Sider hard disk drive (D. E. Hall), Jan 319
on tape backup units (M. C. Rubel), Oct 243
on Toshiba P321 dot-matrix printer (R. D. Swearengin), Nov 255
on Turner Hall Card for IBM PC memory expansion (J. Angel), Sept 287

Hayes Microcomputer Products
Smartmodem 1200, Dec 255

Smartmodem 2400, June 339, Dec 255
Hearing-impaired persons, Kurzweil Voice Writer for (R. Kurzweil), Mar 177
Henon mapping, Dec 170
with Pascal (G. Hughes), Dec 161
Hercules Graphics Card Plus, review (R. Malloy), Dec 249
Hermes real-time multiprocessing system, Modula-2 in development of (R. C. Corbeil, A. H. Anderson), May 111
HERMIES robot, Jan 233
Hewlett-Packard in Japan, Vectra-D Dual-Mode Workstation from (W. M. Raike), Sept 351
High-Performance Systems Stella program (E. Shapiro), May 335
review (S. B. Robinson), Dec 277
Hilbert curve program, in BASIC (M. Ackerman), June 137
HIPAD digitizer for data entry, review (E. D. Hearn), Nov 261
HiSoft DevpacST 68000 assembly language development package (B. Webster), Nov 330
Hochstrasser Computing AG, Modula-2 System for Z80 CP/M from, review (B. R. Anderson), Mar 225
Holland bit-mapped classifier (P. W. Frey), Nov 161
on house architecture, in BASIC, Nov 166
on political predictions, in Pascal (P. A. Schrodt), Nov 177
Home security system, overzealous (S. Ciarcia), Apr 85
Homebound computing in aging, with robotic aids (K. G. Engelhardt, R. Edwards), Mar 191
in blindness, factors considered in (A. Arditi, A. E. Gillman), Mar 199
with Electronic University Network (D. Osgood), Mar 171
in hearing impairment, with Kurzweil Voice Writer (R. Kurzweil), Mar 177
as issue theme (J. M. Tazelaar), Mar 153
in speech impairment, semantic compaction system in (B. R. Baker), Mar 160
and telecommuting (J. M. Tazelaar), Mar 155
Houston Instrument Corp. digitizer for data entry, review (E. D. Hearn), Nov 261
Huffman coding for data compression (J. Amsterdam), May 99
HyperDrive 2000 computer from General Computer Company (B. Webster), Nov 323

I

IBM
best of BIX on, Feb 378, Mar 384, Apr 371, May 405, June 383
personal computer from. *See* IBM PC
Professional Debug Facility from, review (J. C. Carden), Apr 249
Wheelprinter E from, review (R. D. Swearengin), July 315
IBM PC
abstract mathematical art with BASIC program for (K. E. Perry), Dec 181
Abundance Forth-based database language for, in public domain (R. Green), Oct 193
accelerator boards for (S. S. Fried), IBM 141
Actor object-oriented programming language for (C. B. Duff), Aug 211
address ordering in (A. R. Miller), IBM 233
Advantage! multifunction board for PC AT, review (T. J. Byers), Jan 327
Arity/Prolog version 3.2 for, review (W. G. Wong), Mar 245
assembly language interrupt routines for (W. J. Claff), IBM 249
Balance of Power game for (E. Shapiro), Dec 322
best of BIX on, July 403, Aug 351, Sept 398, Oct 333, Nov 390, Dec 362, IBM 288
book reviews related to, IBM 24
Boxes and Arrows program for, May 335
BYSO LISP version 1.17 for, review (W. G. Wong), July 293
CGEN program for translating BASIC into C for (D. Pountain), Oct 311
circuit simulation programs for, July 170
COLOR3D.BAS program for displaying molecules in color with (J. J. Farrell), Feb 149
and compatible products. *See* IBM PC-compatible products
Convertible model, description of (G. M. Vose), IBM 83
Datran Modem Accelerator version 1.1 for, review (B. Nance), Aug 289
Definicon Systems 68020 coprocessor board for (T. Marshall, C. Jones, S. Kluger), July 120, Aug 108
Enable integrated software package for, review (S. King), Jan 331
Enhanced Console Driver for user-friendly interface to DOS of (A. Zackin), Oct 183
filePro 16 database management software for, review (R. Harkness), Nov 297
Flash-Com electronic mail and telecommunications software for, review (B. N. Meeks), Dec 281
GEM Draw painting program for, review (R. Birmele), May 269
graphics with, best of BIX on, Dec 385
graphing quadric surfaces on, with BASIC program (G. Haroney), Dec 215
Guru integrated software package for, review (E. R. Tello), Aug 281
Hardcard hard disk for, review, (E. White), May 273
Henon mapping with Turbo Pascal program for (G. Hughes), Dec 161
Hercules Graphics Card Plus for, review (R. Malloy), Dec 249
increasing speed of PC AT (B. K. Roemmele), IBM 209
benchmarks on (E. White), IBM 212
Induce program for extracting knowledge from data for, in Turbo Pascal, Nov 158
input devices for, non-keyboard, review (C. H. Pappas), May 285
Intuitive Solution business application generator for (D. Pountain), May 363
ITC Modula-2 version for, review (M. Bridger), Oct 255
Japanese-language word processing on (W. M. Raike), Aug 330
Kaypro 286i compared to PC AT (H. Krause), Mar 217
keyed file access method for, in BASIC (S. C. Perry), Sept 137
Lattice 8086/8088 C compiler version 2.15 for, review (D. S. Woolston), Feb 273
Lettrix resident print processor for, review (A. R. Miller), May 299
Lotus Manuscript word processor for, preview (G. A. Stewart), Nov 91
MacView program for transferring Macintosh graphics to (M. Anacker), June 131
Mandelbrot program in Turbo Pascal for (D. Pountain), Sept 359
memory of adjusting size of (R. Miller), IBM 243
AST Enhanced Expanded Memory Specification for expansion of, IBM 170
Lotus/Intel/Microsoft Expanded Memory Specification for expansion of (R. Duncan), IBM 169

manipulation of (A. R. Miller), IBM 233, 243
memory management unit of RT PC (R. S. Simpson), IBM 143
Turner Hall Card for expansion of, review (J. Angel), Sept 287
Microsoft Word version 3.0 word processor for, review (M. C. Rubel), Oct 261
microTSP time-series analysis program version 4.1 for, Apr 257
and MIDI interface hardware construction and software for (J. Kubicky), June 199
hardware reviews (R. Powell, R. Grehan), June 265
Turbo Pascal software for (D. Swearingen), June 211
Mix C compiler for, review (R. Grehan), June 257
muLISP-86 programming language for, review (R. J. Schalkoff), Oct 249
NewWord 3 word processor for, review (J. Heilborn, N. Reel), Aug 273
Norton Utilities, PC Tools and Super Utility software for, comparison of (R. Rabinovitz), Oct 265
Paradox 1.1 relational database program for, review (R. DeMaria), Sept 303
Pascal packages for, comparison of (N. C. Shammas), Dec 265
PC-DOS DEBUG program and BREAKPT routine for (E. Batutis), Sept 127
PCTEX and MicroTEX typesetting packages for, review (H. R. Varian), Apr 267
Pocket APL for, review (E. H. Johnson), Mar 237
Professional Debug Facility and Advanced Fullscreen Debug programs for, review (J. C. Carden), Apr 249
protected mode program for PC AT (R. P. Nelson), IBM 123
Racter program for, review (H. Kenner), May 289
RAM-loadable character sets for (R. Wilton), IBM 197
Reflex analytical database program for, review (R. DeMaria), Aug 277
with RT processor (R. O. Simpson), IBM 43
Software Carousel virtual memory manager for, review (M. Haas), Sept 299
SPSS/PC+ analytical software package for, review (J. M. Jacques), Nov 279
Strike spelling checker for, review (R. Ramsey), Nov 289
structural analysis procedures on with finite-element technique (R. W. Johnson, F. G. Loygorri), July 199
with Truss2 bridge-truss analysis program (C. Pedicini), July 145
tape backup units for, review (M. C. Rubel), Oct 243
three-dimensional graphics program for, in BASIC, Jan 153
for molecules in color, Feb 149
on quadric surfaces, Dec 215
TOPSI 2.0 programming language for, review (L. Moskowitz), Aug 261
Turbo Lightning spelling checker for, review (R. Ramsey), Nov 289
Turbo Prolog programming language for (B. Webster), Sept 335
review (N. C. Shammas), Sept 293
Waltz LISP version 5.01 for, review (W. G. Wong), July 293
Watson deductive reasoning program in Prolog for, Nov 214
WordPerfect 4.1 word processor for, review (R. Birmele), Sept 311
Z80MU program for emulation of Zilog Z80 and CP/M 2.2 for (R. A. Baumann), Oct 203

IBM PC–compatible products
Apricot XEN computer, Apr 311
best of BIX on, June 383, July 403, Aug 361, Sept 398, Oct 333, Nov 390, Dec 362, IBM 288
Canon A-200 computer system, review (P. V. Callamaras), Jan 293
Compaq Deskpro 286 computer, review (S. Miastkowski), June 243
Compaq Deskpro 386 computer, preview (T. Thompson, D. Allen), Nov 84
Compaq Portable II computer, review on Model 3 (S. Miastkowski), Oct 239
Conquest Turbo PC computer, review (J. D. Unger), July 289
developing software for (J. Rosenblum, D. Jacobs), IBM 181
Epson Equity I computer, review (J. D. Unger), Nov 239
Epson Equity III computer, review (W. Rash Jr.), Dec 239
ITT XTRA XP computer, review (J. D. Unger), July 281
Kaypro PC, review (J. D. Unger), Nov 239
Leading Edge Model D PC computer, review (S. Miastkowski), Sept 269
MacCharlie accessory for Macintosh providing access to (L. Crockett), Feb 262
Multitech MPF-PC/700 D1 computer, review (J. D. Unger), Nov 239
NCR PC6 computer, review (A. Little), Aug 241
Panasonic Exec. Partner transportable computer, review (R. Malloy), Apr 231
Scottsdale Systems Color Fox computer system, review (J. D. Unger), Jan 301
Sperry PC/IT, review (F. D. Davis), Aug 247
at Taiwan Computex show of 1986 (W. M. Raike), Oct 307
TeleVideo Tele-286 Model 2 computer, review (W. Rash Jr.), June 251
Western AT computer, review (W. Rash Jr.), Dec 239
Xerox 6060 computer, review (W. Rash Jr.), Sept 275
Zenith Z-241 computer, review (W. Rash Jr.), Dec 239
Zenith Z-248 computer, review (W. Rash Jr.), Dec 239

Icon high-level programming language (R. E. Griswold, M. T. Griswold), Oct 167

Iconic user interface
in object-oriented programming (B. Cox, B. Hunt), Aug 161
performance of (J. Uebbing, C. Young), Aug 176
in semantic compaction system (B. R. Baker), Mar 160

ID3 algorithm for extracting knowledge from data (B. Thompson, W. Thompson), Nov 149

ifCOM7 laptop portable computer from Oki (W. M. Raike), July 371

ILS laboratory interfacing software package, July 303

Image-processing
technology
glossary on, Mar 110
in locating *Titanic* (M. Spalding, B. Dawson), Mar 97

Images
in iconic user interface, Aug 161, 176
in semantic compaction system (B. R. Baker), Mar 160

Images painting program for Amiga, review (W. Block), Nov 285

INCOMM Data Systems Rainbow 1200SA modem, Dec 255

Indexes
on magazine and newspaper articles, on optical disks, May 236
on records in files
in keyed access method, in BASIC, Sept 138

programming project on, in Pascal (B. Webster), June 93

Information Access Company Info Tract magazine and newspaper index (J. Dorner), May 236

InfoStructures PopDrop program (E. Shapiro), Nov 362

Info Tract magazine and newspaper index (J. Dorner), May 236

Inmos Transputer (D. Pountain), July 363
Meiko Computing Surface based on (D. Pountain), July 364

Inner Loop Software Boxes and Arrows program (E. Shapiro), May 335

Input comparison using 2ⁿ property, programming insight on (R. C. Arp Jr.), Oct 145

Input devices, non-keyboard, review (C. H. Pappas), May 285

Instant Music program for Amiga (B. Webster), Dec 308

Integral approximation with Aitken extrapolation (D. M. Smith), Dec 113

Integrated software packages
Enable
version 1.0 (S. King), Jan 331
version 1.1 (R. Malloy), Jan 334
Guru, review (E. R. Tello), Aug 281
Works, for Macintosh (E. Shapiro), Nov 365

Intel
8031 processor, construction of emulator board for in-circuit emulation of (G. Dinwiddie), July 181
80286 processor, IBM 112
in Apricot XEN computer, Apr 305
in IBM PC AT, IBM 112, 123
in Kaypro 286i computer, Mar 217
in micro-based supercomputer (N. H. Christ, A. E. Terrano), Apr 145
program using native code of (R. P. Nelson), IBM 123
protected mode of, IBM 112, 123
80287 processor, in micro-based supercomputer (N. H. Christ, A. E. Terrano), Apr 145
80386 processor, Nov 88
architecture of (P. Wells), IBM 89
Compaq Deskpro 386 computer based on, Nov 84
paging with, IBM 89, 111
segmentation with, IBM 89, 111
systems implications of (B. Nicholls), IBM 98
virtual memory with, IBM 89, 111
Lotus/Intel/Microsoft Expanded Memory Specification (R. Duncan), IBM 169

Intelligent Assistant of Q&A software package, Jan 120

Intelligent Digitizers, review (E. D. Hearn), Nov 261

Intelligent serial EPROM programmer, construction of (S. Ciarcia), Oct 103

Interface Technologies Corporation Modula-2 Software Development System, review (M. Bridger), Oct 255

Interfacing
laboratory interfacing software packages, review (P. Wirth, L. E. Ford), July 303
machine language routine to Pascal (J. Feldman), Sept 145
MIDI (musical instrument digital interface) specifications in. *See* MIDI specifications
parallel
applications of (S. Ciarcia), Aug 97
introduction to (S. Ciarcia), July 85
with Printit serial/parallel printer interface card (H. Brugsch, J. J. Lazzaro), Mar 261
with user. *See* User interface

Interlace database program (E. Shapiro), May 336

Interrupt routines for IBM PC
in assembly language (W. J. Claff), IBM 249
in BREAKPT routine for PC-DOS DEBUG program (E. Batutis), Sept 127

Intuitive Solution business application generator (D. Pountain), May 363

Invention Software Pascal Extender program for Macintosh (B. Webster), July 350

Inversion of large matrices, with Pan and Reif algorithm (T. E. Phipps Jr.), Apr 181

Iomega Corporation Mac Bernoulli Box, Feb 344

ISM Surgeon surgery simulation program (E. Shapiro), Dec 322

It Figures spreadsheet program (E. Shapiro), July 381

ITC Modula-2, review (M. Bridger), Oct 255

ITT XTRA XP computer, review (J. D. Unger), July 281

J

Japan, BYTE reports from (W. M. Raike)
on coin-operated database service, Nov 347
on COMDEX show of 1986 in Japan, July 371
on Data Show of 1985, Feb 317
on DeSmet C development package, June 339
on DynaMac Japanese version of Macintosh, Jan 382
on EM/3+ software from Megasoft, May 329
on FM-16π computer upgrade, Aug 329
on FM-16π portable computer, Jan 384 Mar 330
on Hayes Smartmodem 2400, June 339
on Japanese-language bulletin board system, Nov 348
on Japanese-language word processing on IBM PC, Aug 330
on Keystyle 80 keyboard and laptop computer, Oct 309
on Mind Japanese programming language, Mar 327
on NEC PC-9801VM2E personal computer, Nov 347
on NEC PC-9801VM4 personal computer, July 372
on NEC upgrades of PC 9801 series, Jan 381
on Oki ifCOM7 laptop portable computer, July 371
on Software Show of 1985, Feb 317
on Taiwan Computex Show of 1986, Oct 307
on Tokyo Microcomputer Show of 1986, Sept 351
on Vectra-D Dual-Mode Workstation, Sept 351
on Yamaha Piano Player, Sept 351

Japanese language bulletin board system in (W. M. Raike), Nov 348
word processing on IBM PC in (W. M. Raike), Aug 330

K

KAMASOFT KAMAS outline processor, review (A. S. Woodhull), Apr 241

Kaypro computers
Kaypro 10, Mix C Compiler for, review (R. Grehan), June 257
Kaypro 286i, review (H. Krause), Mar 217
Kaypro PC, review (J. D. Unger), Nov 239

KeepTrack Plus software (E. Shapiro), June 334

Keyboards
efficiency of Dvorak and QWERTY layouts (D. W. Olson, L. E. Jasinski), Feb 241
Keystyle 80 keyboard and laptop computer (W. M. Raike), Oct 309

Keyed file access in BASIC (S. C. Perry), Sept 137

Keystyle 80 keyboard and laptop computer (W. M. Raike), Oct 309

Knowledge representation
in bit-mapped classifier system (P. W. Frey), Nov 161
on house architecture, in BASIC, Nov 166
on political predictions, in Pascal (P. A. Schrodt), Nov 177
handling data conflicts in, with decision support system in Prolog (C. Y. Cuadrado, J. L. Cuadrado), Nov 193
ID3 algorithm for extracting knowledge from data (B. Thompson, W. Thompson), Nov 149
as issue theme (C. Baskin), Nov 147
in Marvin program for teaching computers to learn, in Prolog (A. T. Kolokouris), Nov 225
in OPS5 (Official Production System version 5) programming language (L. Moskowitz), Nov 217
in Watson program for deductive reasoning, in Prolog (J.-C. Emond, A. Paulissen), Nov 207

Knuth, Donald, interview with (G. M. Vose, G. Williams), Feb 169

Kriya Systems Neon object-oriented language for Macintosh, Aug 179

Kurzweil
250 Digital Synthesizer, review (C. Morgan), June 279
Voice Writer (R. Kurzweil), Mar 177

Kyocera International
KM1200S modem, Dec 255

L

Laboratories
LabView Laboratory Virtual Instrument Engineering Workbench programming environment for, preview (G. M. Vose, G. Williams), Sept 84
robotic automation of, for organic chemistry research (G. W. Kramer, P. L. Fuchs), Jan 263
software interfacing packages for, review (P. Wirth, L. E. Ford), July 303

Laboratory Technologies
Labtech Notebook software, July 303

LABPAC laboratory interfacing software package, July 303

Labtech Notebook
laboratory interfacing software package, July 303

LabView Laboratory Virtual Instrument Engineering Workbench, preview (G. M. Vose, G. Williams), Sept 84

Languages
Abundance FORTH-based database language, in public domain (R. Green), Oct 193
Actor object-oriented programming language (C. B. Duff), Aug 211
Ada, best of BIX on, Nov 405
assembly. *See* Assembly language
BASIC. *See* BASIC
C. *See* C language
FORTH. *See* FORTH
FORTRAN. *See* FORTRAN
functional, data-flow in (C. Hankin, D. Till, H. Glaser), May 123
Icon high-level programming language (R. E. Griswold, M. T. Griswold), Oct 167
interfacing machine language routine to Pascal (J. Feldman), Sept 145
of KAMAS outline processor, Apr 241
LISP. *See* LISP language
Mind Japanese programming language (W. M. Raike), Mar 327
Modula-2. *See* Modula-2
natural
parallel interpretation of (J. Pollack, D. L. Waltz), Feb 189
of Q&A software package, Jan 120
in object-oriented programming
on Macintosh (K. J. Schmucker), Aug 177
with Smalltalk (T. Kaehler, D. Patterson), Aug 145
Occam (D. Pountain), July 363, Sept 359
dynamic load balancing in (D. Pountain), July 368, Sept 359
OPS5 (Official Production System version 5) programming language (L. Moskowitz), Nov 217
AUTO.OPS implementation of, Nov 222
ExperOPS5 implementation of, for Macintosh (W. Jacobs), July 297
TOPSI 2.0 implementation of, for IBM PC (L. Moskowitz), Aug 261
PAL (Paradox Application Language), Sept 303
Pascal. *See* Pascal
Pocket APL, review (E. H. Johnson), Mar 237
Prolog. *See* Prolog
SIMPL (J. Amsterdam), Jan 131, Feb 103
SNOBOL4, processing strings in (J. F. Gimpel), Feb 175

Lap-size portable computers. *See* Portable computers

Laser disks, databases available on (N. Desmarais), May 235

Laser printers, LaserWriter Plus, June 88

LaserWriter Plus laser printer, June 88

Lattice Inc.
Amiga C compiler from, review (C. Heath), Nov 271
Mandelbrot program using (P. B. Schroeder), Dec 207
IBM PC C compiler from, review (D. S. Woolston), Feb 273
Screen Editor program editor for Amiga from (B. Webster), July 352

Leading Edge Model D PC computer, review (S. Miastkowski), Sept 269

Let's C and csd C development package and symbolic debugger, review (W. G. Wong), Aug 267

Lettrix resident print processor, review (A. R. Miller), May 299

Levco Prodigy 4 computer (B. Webster), Nov 323

Levien BYSO LISP version 1.17, review (W. Wong), July 293

Lifetree Software
Volkswriter 3 program (E. Shapiro), June 332

Light Pen input device, review (C. H. Pappas), May 285

Lightspeed C development environment for Macintosh (B. Webster), Aug 323, Sept 340

LISP language
BYSO LISP version 1.17 implementation of, review (W. Wong), July 293
EDITNET program in, Feb 196
muLISP-86 LISP development system, review (R. J. Schalkoff), Oct 249
Visual Syntax editor for programming in (R. Levien), Feb 135
Waltz LISP version 5.01 implementation of, review (W. Wong), July 293

Living Videotext
More outline processor for Macintosh from, Sept 367, Dec 316
Ready! memory-resident outline processor from, Mar 297

Logitech LogiMouse C7 (E. Shapiro), Dec 324

Lotus Development Corporation
Lotus/Intel/Microsoft Expanded Memory Specification (R. Duncan), IBM 169
Manuscript word processor from, preview (G. A. Stewart), Nov 91

M

MacAPP object-oriented application framework for Macintosh (K. J. Schmucker), Aug 189
programming experience with, Aug 200

MacBottom Macintosh hard disk (B. Webster), June 348

Mac C C compiler version 4.01 for Macintosh (B. Webster), Apr 318

MacCharlie Macintosh accessory from Dayna Communications, review (L. Crockett), Feb 262

MacDraw painting program, review (R. Birmele), May 269

MacFORTH, DRAGON program in (B. R. Land), Apr 137

Machine language routine, interfaced to Pascal (J. Feldman), Sept 145

Machine learning with Marvin program in Prolog (A. T. Kolokouris), Nov 225

Macintosh computer
512K-byte version of (B. Webster), Nov 323
Acta outline processor for (B. Webster), Dec 316
Bernoulli Box for (B. Webster), Feb 344
best of BIX on, Feb 388, Mar 392, Apr 376, May 414, June 387, July 407, Aug 354, Sept 404, Oct 335, Nov 394, Dec 370
ComicWorks painting program for (E. Shapiro), Dec 321
compared to Atari and Amiga computers (B. Webster), Mar 305, May 343
ConcertWare+ and SongPainter programs for, review (M. S. Bernardo), June 273
Dark Castle game for (E. Shapiro), Dec 327
DASCH external RAM disk for (B. Webster), June 353
debugging programs on (J. West), Dec 127
DeSmet C Development Package for, review (J. Robie), Aug 253
digital music synthesis on (C. Yavelow), June 171
with Kurzweil 250 Digital Synthesizer (C. Morgan), June 279
DRAGON program for, in Forth (B. R. Land), Apr 137
DynaMac Japanese version of, Jan 382
ExperOPS5 expert-system programming language for, review (W. Jacobs), July 297
Explorer disassembler program for, programming of (O. Andrade), Mar 135
fonts for (B. Webster), Feb 340
Interlace database program for (E. Shapiro), May 336
LabView Laboratory Virtual Instrument Engineering Workbench for, preview (G. M. Vose, G. Williams), Sept 84
Lightspeed C development environment for (B. Webster), Aug 323, Sept 340
MacApp object-oriented application framework for (K. J. Schmucker), Aug 189
programming experience with, Aug 200
MacBottom hard disk for (B. Webster), June 348
Mac C C compiler version 4.01 for (B. Webster), Apr 318
MacCharlie accessory for, review (L. Crockett), Feb 262
MacDraw painting program for, review (R. Birmele), May 269
MacPaint program for graphics on, transferred to IBM PC with MacView program (M. Anacker), June 131
Mapper similarity mapping program for, in BASIC (R. Spencer), Aug 85
mass storage devices for, benchmark tests on, June 354
MIDI interfaces for, hardware reviews (R. Powell, R. Grehan), June 265
More outline processor for, Sept 367, Dec 316
object-oriented languages for (K. J. Schmucker), Aug 177
Pascal compilers for (B. Webster), Feb 340, Apr 315
Pascal Extender software for (B. Webster), July 350
Plus version of (B. Webster), Nov 323
compared to Atari and Amiga computers (B. Webster), May 343
description of (P. Robinson), June 85
memory expansion kits for, comparison of (C. Crawford, T. Thompson), Nov 250
system review (C. Crawford), Nov 247
spreadsheet program in Modula-2 for, programming project on (J. Amsterdam), July 97
Stella program for, May 335
review (S. B. Robinson), Dec 277
Surgeon surgery simulation program for (E. Shapiro), Dec 322
system calls of, compared to Amiga (A. B. Webber), Sept 249
Talking Moose public domain desk accessory for (B. Webster), Dec 310
three-dimensional graphics program for, in BASIC, Jan 153
Turbo Pascal for (B. Webster), Nov 338
Works integrated software package for (E. Shapiro), Nov 365

MacLanguage Series Pascal compiler from TML Systems (B. Webster), Feb 340, Apr 315

Macmillan Software ASYST program, July 303

MacPaint program for Macintosh graphics, transferred to IBM PC with MacView program (M. Anacker), June 131

MacView program, for transferring Macintosh graphics to IBM PC (M. Anacker), June 131

Magazine index on optical disks, May 326

Magnetic disks compared to CD-ROM (B. Zoellick), May 177
and tape backup systems (A. Antonuccio), May 227

Mail, electronic, with Flash-Com electronic mail and telecommunications software system, review (B. N. Meeks), Dec 281

Mandelbrot program for Amiga in Lattice C (P. B. Schroeder), Dec 207
for IBM PC in Turbo Pascal (D. Pountain), Dec 359

Manuscript word processor from Lotus, preview (G. A. Stewart), Nov 91

Manx Aztec C68K C compiler for Amiga, review (C. Heath), Nov 271

Mapper similarity mapping program for Macintosh, in BASIC (R. Spencer), Aug 85

Mapping
Henon, Dec 170
with Pascal (G. Hughes), Dec 161
similarity, with Mapper program for Macintosh (R. Spencer), Aug 85

Mark Williams Company Let's C and csd C development package and symbolic debugger, review (W. G. Wong), Aug 267

Marvin program in Prolog, teaching computers to learn (A. T. Kolokouris), Nov 225

MAS-20 hard disk drive for Amiga (B. Webster), Dec 306

Mass storage
and CD-ROM software development (B. Zoellick), May 177
evolution of (L. Laub), May 161
as issue theme (K. Sheldon), May 159
on optical disks. *See* Optical disks
and tape backup systems (A. Antonuccio), May 227

hardware reviews (M. C. Rubel), Oct 243

Material selection in construction
with MSP program in Turbo Pascal (T. Sawyer, M. Pecht), July 235
with physical property estimation programs in BASIC (J. N. Stone), July 253

Mathematics
abstract mathematical computer graphics with BASIC program (K. E. Perry), Dec 181
approximating integrals with Aitken extrapolation (D. M. Smith), Dec 113
ATOMCC toolbox method for solving differential equations, in FORTRAN (Y. F. Chang), Apr 215
computer approximations in, Apr 191, 196, 216, Dec 113
Chebyshev, Apr 174
speed and precision of (S. L. Moshier), Apr 161
conversion of repeating decimals to fractions (R. T. Kurosaka), Jan 397
in cyclic redundancy check calculations (G. Morse), Sept 115
Diophantine equations in (R. T. Kurosaka), Mar 343
Euclid's algorithm in (R. T. Kurosaka), Jan 397, Mar 343
extracting *n*th root from binary numbers (L. S. Wo), Nov 115
Henon mapping with Pascal (G. Hughes), Dec 161
laws of, in programming (C. A. R. Hoare), Aug 115
with Mandelbrot program for Amiga in Lattice C (P. B. Schroeder), Dec 207
for IBM PC in Turbo Pascal (D. Pountain), Sept 359
with micro-based supercomputer (N. H. Christ, A. E. Terrano), Apr 145
and musical fractals (C. Dodge, C. R. Bahn), June 185
and number crunching as issue theme (T. Clune), Apr 143
number-sequence games in BASIC (R. T. Kurosaka), Aug 333
Pan and Reif algorithm for inversion of large matrices (T. E. Phipps Jr.), Apr 181
Pellian equation in (R. T. Kurosaka), May 379
plotting equations in three dimensions, with BASIC program (G. Haroney), Dec 215
polar method for generating normal deviates (A. Latour), Aug 131
probability analysis programs on sucker bets (R. T. Kurosaka), Nov 373
Runge-Kutta method for solving differential equations, Apr 216
in BASIC (D. M. Leo), Apr 196
in FORTRAN (B. Thomas), Apr 191
time-series analysis in, Apr 262
with microTSP program version 4.1 (P. Davenport), Apr 257
whole-number solutions to equations in (R. T. Kurosaka), Mar 343

Matrices, large, Pan and Reif algorithm for inversion of (T. E. Phipps Jr.), Apr 181

Megamax C development system for Atari ST (B. Webster), Nov 326

Megasoft EM/3+ software, May 329

Meiko Computing Surface (D. Pountain), July 364

Memory
Access Associates Alegra Memory Expansion Box for Amiga (B. Webster), Dec 305
Amiga ROM Kernel, introduction to (R. J. Mical), Feb 116
compact disk-read only memory. *See* CD-ROM
DASCH external RAM disk for Macintosh (B. Webster), June 353
of EPROM programmers. *See* EPROM programmers
of IBM PC
adjusting size of (A. R. Miller), IBM 243
AST Enhanced Expanded Memory Specification for expansion of, IBM 170
Lotus/Intel/Microsoft Expanded Memory Specification for expansion of (R. Duncan), IBM 169
manipulation of (A. R. Miller), IBM 233, 243
memory management unit of RT PC (R. S. Simpson), IBM 143
Turner Hall Card for expansion of, review (J. Angel), Sept 287
of Macintosh Plus, memory expansion kits for (C. Crawford, T. Thompson), Nov 250
mass storage as issue theme (K. Sheldon), May 159. *See also* Mass storage
Motorola 68000 architecture memory management units (G. Zehr), Nov 127
packed-pixel organization of, Dec 200
programming of storage allocators in Modula-2 (J. Amsterdam), Oct 123
RAM-loadable character sets for IBM PC (R. Wilton), IBM 197
virtual (J. Shiell), IBM 111
with Intel 80386 CPU, IBM 89, 111
Software Carousel virtual memory manager (M. Haas), Sept 299

Memory management units
in IBM RT PC (R. S. Simpson), IBM 43
for Motorola 68000 family of processors (G. Zehr), Nov 127

Message passing procedures in Modula-2, May 115

Metacomco Tripos operating system (D. Pountain), Feb 321

Metadigm Metascope debugger (B. Webster), Nov 334

METAFONT typesetting system, Feb 169, 211

Metascope debugger for Amiga (B. Webster), Nov 334

MetaWare Professional Pascal version 2.5, review (N. C. Shammas), Dec 265

Methods version of Smalltalk language, programming experience with, Aug 206

MEX114.COM public domain communications program, Oct 220

MichTron Atari 520ST software (B. Webster), Apr 318

MicroBotics MAS-20 hard disk drive for Amiga (B. Webster), Dec 306

Micro Data Base Systems Guru integrated software package, review (E. R. Tello), Aug 281

Microelectronic technology in Soviet Union (P. Walton), Nov 137

Microgrid digitizers, review (E. D. Hearn), Nov 261

MicroPro International
Easy word processor from, Mar 300
WordStar 2000 Release 2 software from (E. Shapiro), June 329

Microsafe series of finite-element analysis programs, July 208

Microshop Computer Products Conquest Turbo PC computer, review (J. D. Unger), July 289

MicroSmiths TxEd program editor for Amiga (B. Webster), Nov 336

Microsoft
BASIC
Macintosh Explorer disassembler program in, Mar 145
Mapper similarity mapping program for Macintosh in (R. Spencer), Aug 85
subroutine overlays in (M. Carmichael), May 151
translated into C, with CGEN program (D. Pountain), Oct 311
Lotus/Intel/Microsoft Expanded Memory Specification (R. Duncan), IBM 169
MS-DOS. *See* MS-DOS
MS-FORTRAN, assembly routines in (M. Dahmke), IBM 217
MS-Pascal version 3.31, review (N. C. Shammas), Dec 265

Windows on Apricot XEN computer, Apr 309
Word 3.0 word processor (E. Shapiro), Aug 322
review (M. C. Rubel), Oct 261
Works integrated software package for Macintosh (E. Shapiro), Nov 365

MicroTEX typesetting package, review (H. R. Varian), Apr 267

MicroTSP time-series regression package, version 4.1, review (P. Davenport), Apr 257

MIDI (musical instrument digital interface)
specifications, June 145, 172, 199
hardware reviews related to (R. Powell, R. Grehan), June 265
for IBM PC
hardware construction and software for (J. Kubicky), June 199
hardware reviews (R. Powell, R. Grehan), June 265
Turbo Pascal software tools for (D. Swearingen), June 211
of Kurzweil 250 Digital Synthesizer, June 279

MIDIMAC MIDI Interface for Macintosh, hardware review (R. Powell, R. Grehan), June 265

Mind Japanese programming language (W. M. Raike), Mar 327

Mindreader word processor (E. Shapiro), Aug 319

Mindscape
Balance of Power game (E. Shapiro), Mar 299, Dec 322
ComicWorks painting program (E. Shapiro), Dec 321
Racter program, review (H. Kenner), May 289

Mining, underground, stress analysis of tunnels in (D. L. Petersen, S. L. Crouch), July 219

Miracle Technology WS3000 modem (D. Pountain), June 319

Mirror telecommunications program (E. Shapiro), Apr 329

Mix C compiler for IBM PC, review (R. Grehan), June 257

MM digitizers for data entry, review (E. D. Hearn), Nov 261

Models. *See* Simulation

Modems
comparison of (S. Satchell), Dec 255
Datran Modem Accelerator version of 1.1 for text transmission, review (B. Nance), Aug 289
Hayes Smartmodem 2400 (W. M. Raike), June 339
Miracle Technology WS3000 modem (D. Pountain), June 319

Modula-2
creating reusable modules in (N. C. Shammas), Jan 145
Hochstrasser system for Z80 CP/M system (B. R. Anderson), Mar 225
Huffman coding programs for data compression in, May 108
from Interface Technologies Corporation, review (M. Bridger), Oct 255
spreadsheet program for Macintosh in, programming project on (J. Amsterdam), July 97
storage allocators in (J. Amsterdam), Oct 123
as systems programming language (R. C. Corbeil, A. H. Anderson), May 111
TSI Modula-2/ST compiler (B. Webster), Feb 332

Modules, reusable, creation of (N. C. Shammas), Jan 145

Modulo 2 division, in calculation of cyclic redundancy checks (G. Morse), Sept 115

Molecules displayed in color
with CHEMMOD molecular modeling system (D. Pountain), Dec 334
with COLOR3D.BAS program for IBM PC (J. J. Farrell), Feb 149

Monitors, red-green-blue, COLOR3D.BAS program for displaying molecules in color on (J. J. Farrell), Feb 149

More outline processor for Macintosh, Sept 367, Dec 316

Morgan Computing Advanced Trace86/ assembler/debugger (B. Webster), Oct 293

Motherboard accelerators for IBM PC, IBM 156

Motion-impaired persons, robotic aids increasing independence of (K. G. Engelhardt, R. Edwards), Mar 191

Motion in robotics
and autonomous robot navigation (C. Jorgensen, W. Hamel, C. Weisbin), Jan 223
and coordination of multiple manipulators (J. S. Hawker, R. N. Nagel, R. Roberts, N. G. Odrey), Jan 203

Motorola
68000 processor
in Amiga computer, Oct 231. *See also* Amiga computer from Commodore
and animation software for Amiga (E. A. Ditton, R. A. Ditton), Sept 241
architecture and Unix compatibility of (A. L. Rood, R. C. Cline, J. A. Brewster), Sept 179
assembly language programming guidelines for (M. Morton), Sept 163
in Atari computers, Jan 84. *See also* Atari computers compared to other 68000-series processors (T. L. Johnson), Sept 205
comparison of Macintosh, Atari, and Amiga computers based on (B. Webster), Mar 305, May 343
HiSoft DevpacST 68000 assembly language development package, Nov 330
as issue theme (G. M. Vose), Sept 161
in Macintosh, Mar 305, May 343. *See also* Macintosh computer
memory management units for (G. Zehr), Nov 127
and sound system of Amiga (D. D. Thiel), Oct 139
and system calls of Amiga and Macintosh (A. B. Webber), Sept 249
and TOS operating system of Atari ST computers (M. Rothman), Sept 223
VME/10 system based on, review (R. E. Robinson III), Feb 253
68008 processor, compared to other 68000-series processors (T. L. Johnson), Sept 205
68010 processor
in AT&T UNIX PC, May 254
compared to other 68000-series processors (T. L. Johnson), Sept 205
of VME/10 system, Feb 253
68012 processor compared to other 68000-series processors (T. L. Johnson), Sept 205
68020 processor
compared to other 68000-series processors (T. L. Johnson), Sept 205
in Definicon 68020 coprocessor (T. Marshall, C. Jones, S. Kluger), July 120, Aug 108
68881 floating-point chip in Definicon 68020 coprocessor, July 120
VME/10 system, review (R. E. Robinson III), Feb 253

Mouse
of AT&T UNIX PC, May 254
Logitech LogiMouse C7 (E. Shapiro), Dec 324

MPF-PC/700 D1 computer, review (J. D. Unger), Nov 239

MPU-401 MIDI interface
for IBM PC
hardware review (R. Powell, R. Grehan), June 265
Turbo Pascal programs for (D. Swearingen), June 211

MS Associates CGEN program for translating BASIC into C (D. Pountain), Oct 311

MS-DOS
Arity/Prolog version 3.2 for, review (W. G. Wong), Mar 245
best of BIX on, IBM 312
comparison of PCTEX and MicroTEX typesetting packages for (H. R. Varian), Apr 267
Eco-C88 C compiler for, review (D. D. Clark), Jan 307
EM/3+ programs running under, May 329
of IBM PC-compatible products, developing software for (J. Rosenblum, D. Jacobs), IBM 181
Mirror telecommunications program for, Apr 329
PD PROLOG public domain implementation of Prolog for (R. Morein), Oct 155

muLISP-86 LISP development system, review (R. J. Schalkoff), Oct 249

Multiprocessing operating systems, May 114
Modula-2 in development of Hermes system (R. C. Corbeil, A. H. Anderson), May 111

Multitasking
with Amiga ROM Kernel, Feb 116
with Intel 80386 CPU (P. Wells), IBM 89

Multitech Electronics
MPF-PC/700 D1 computer, review (J. D. Unger), Nov 239

Multi-Tech Systems
modems, Dec 255

Music, computer
on Amiga (D. D. Thiel), Oct 139
with Instant Music program (B. Webster), Dec 308
book reviews related to, June 63
digital synthesizers in, overview on (R. A. Moog), June 155
fractals in (C. Dodge, C. R. Bahn), June 185
on IBM PC and MIDI interface
hardware construction project and software for (J. Kubicky), June 199
hardware reviews (R. Powell, R. Grehan), June 265
Turbo Pascal software tools for (D. Swearingen), June 211
as issue theme (R. Grehan), June 143
with Kurzweil 250 Digital Synthesizer, review (C. Morgan), June 279
on Macintosh
with ConcertWare+ and SongPainter programs (M. S. Bernardo), June 273
with digital music synthesis (C. Yavelow), June 171
with MIDI interfaces, hardware reviews (R. Powell, R. Grehan), June 265
MIDI specifications in. *See* MIDI specifications
software for, overview on (R. Powell), June 145
with Yamaha Piano Player (W. M. Raike), Sept 351

N

National Instruments
LabView Laboratory Virtual Instrument Engineering Workbench, preview (G. M. Vose, G. Williams), Sept 84

Natural language
parallel interpretation of (J. Pollack, D. L. Waltz), Feb 189
of Q&A software package, Jan 120

Navigation of robots,
autonomous (C. Jorgensen, W. Hamel, C. Weisbin), Jan 223

NCR
5380 SCSI bus-interface chip, added to SB180 computer (S. Ciarcia), May 85, June 107
PC6 computer, review (A. Little), Aug 241

NEC
PC-9801VM2E personal computer (W. M. Raike), Nov 347
PC-9801VM4 personal computer (W. M. Raike), July 372
upgrades of PC-9801 series (W. M. Raike), Jan 381

Neon object-oriented language for Macintosh, Aug 179

NewStar Software
NewWord 3 word processor, review (J. Heilborn, N. Reel), Aug 273

New Sweep public domain file maintenance program, Oct 219

NewWord 3 word processor, review (J. Heilborn, N. Reel), Aug 273

Newspaper index on optical disks, May 236

Nippon. *See* NEC

Nissho NP-2410 24-pin dot-matrix printer, review (R. D. Swearengin), Nov 255

Norton Utilities version 3.1, review (R. Rabinovitz), Oct 265

Novation Smart-CAT Plus modem, Dec 255

Nuclear weapons, BASIC programs calculating impact of (J. R. Fanchi), Dec 143

NULU151.COM public domain library utility, Oct 220

Numbers. *See* Mathematics

Numonics digitizers for data entry, review (E. D. Hearn), Nov 261

O

Object Assembler object-oriented language for Macintosh, Aug 182

Object Logo object-oriented language for Macintosh, Aug 184

Object-oriented programming
Actor language in (C. B. Duff), Aug 211
best of BIX on, Aug 357
data structures in, Aug 142
elements of (G. A. Pascoe), Aug 139
in Forth (D. Pountain), Aug 227
queue simulation program, Aug 230
iconic user interfaces in (B. Cox, B. Hunt), Aug 161
performance of (J. Uebbing, C. Young), Aug 176
improving efficiency of (C. B. Duff), Aug 211
interviews with programmers on (L. Tesler), Aug 195
as issue theme (E. White, R. Malloy), Aug 137
with MacApp application framework (K. J. Schmucker), Aug 189
on Macintosh (K. J. Schmucker), Aug 177
pseudovariables in, Aug 144
software-ICs in, reusable (B. Cox, B. Hunt), Aug 161
syntax and design of Smalltalk language for (T. Kaehler, D. Patterson), Aug 145

Object Pascal object-oriented language for Macintosh, Aug 177

Objective-C object-oriented language for Macintosh, Aug 182
programming experience with, Aug 202

Occam language (D. Pountain), July 363, Sept 359
dynamic load balancing in (D. Pountain), July 368, Sept 359

Oki ifCOM7 laptop portable computer (W. R. Raike), July 371

Omni Computer Systems
Flash-Com electronic mail and telecommunications software system, review (B. N. Meeks), Dec 281

OmniTel 1200 modem, Dec 255

Opcode Systems
MIDIMAC MIDI interface for,

Powell, R. Grehan), June 265

Operating systems
AmigaDOS, development of (D. Pountain), Feb 321
Apple ProDOS, catalog sort routine for (A. C. Silvestri), June 117
CP/M. *See* CP/M operating systems
of Definicon 68020 coprocessor (T. Marshall, C. Jones, S. Kluger), July 120
EM/3+ operating system unification adapter, May 329
MS-DOS. *See* MS-DOS
multiprocessing, May 114
Modula-2 in development of Hermes system (R. C. Corbeil, A. H. Anderson), May 111
PC-DOS
DEBUG program and BREAKPT routine for (E. Batutis), Sept 127
PD PROLOG public domain implementation of Prolog for (R. Morein), Oct 155
Professional Debug Facility and Advanced Fullscreen Debug programs for, review (J. C. Carden), Apr 249
TOS operating system of Atari ST computers (M. Rothman), Sept 223
Tripos system from Metacomco (D. Pountain), Feb 321
UNIX
compatibility with Motorola MC68000 architecture (A. L. Rood, R. C. Cline, J. A. Brewster), Sept 179
security features of (A. Filipski, J. Hanko), Apr 113
System V version 2, in AT&T UNIX PC, May 254
user-friendly interface to, with Enhanced Console Driver (A. Zackin), Oct 183
using DOS functions from Turbo Pascal (D. F. Yriart), Dec 103

OPS5 (Official Production System version 5) programming language (L. Moskowitz), Nov 217
AUTO.OPS implementation of, Nov 222
ExperOPS5 implementation of, for Macintosh (W. Jacobs), July 297

Optical disks, May 218
development of, May 162
error correction in, May 198
codes used in (S. W. Golomb), May 203
Info Tract magazine and newspaper index on (J. Droner), May 236
new products related to (R. Malloy), May 215
read-only, databases available on (N. Desmarais), May 235
write-once, May 215
programming with (J. R. Dulude), May 193

Optimized Systems Software Personal Pascal for Atari ST computers (B. Webster), July 347

Orpheus program for poetry processing, Feb 221

Outline processors
Act, for Macintosh (B. Webster), Dec 316
KAMAS, review (A. S. Woodhull), Apr 241
More, for Macintosh, Sept 367, Dec 316
PC-Outline (E. Shapiro), May 336
Ready!, Mar 297

P

Packed-pixel memory organization, Dec 200

Paging with Intel 80386 CPU, IBM 89, 111

Painting programs
Aegis Images program for Amiga, review (W. Block), Nov 285
ComicWorks program for Macintosh (E. Shapiro), Dec 321
comparison of GEM Draw and MacDraw programs (R. Birmele), May 269
FullPaint program for Macintosh, transferred to IBM PC with MacView program, June 131

PAL (Paradox Application Language) programming language, Sept 303

Pan and Reif algorithm for inversion of large matrices (T. E. Phipps Jr.), Apr 181

Panasonic Exec. Partner transportable computer, review (R. Malloy), Apr 231

Paradox Application Language, Sept 303

Paradox 1.1 relational database program, review (R. DeMaria), Sept 303

Parallel processing
applications of (S. Ciarcia), Aug 97
dynamic load balancing in (D. Pountain), July 368, Sept 359
introduction to (S. Ciarcia), July 85

Parameter passing, with assembly routines in MS-FORTRAN programs (M. Dahmke), IBM 217

Pascal
anagram solving in (E. Keefer), July 113
best of BIX on, Sept 409, IBM 288
compared to ITC Modula-2 (M. Bridger), Oct 255
file-indexing program in, June 96
Henon mapping with (G. Hughes), Dec 161
Holland classifier in, on political predictions, Nov 188
for IBM PC, comparison of versions of (N. C. Shammas), Dec 265
Induce program for extracting knowledge from data in, for IBM PC, Nov 158
interfacing machine language routine to (J. Feldman), Sept 145
MacLanguage Series Pascal compiler for Macintosh (B. Webster), Feb 340, Apr 315
MacView program for transferring Macintosh graphics to IBM PC in, June 131
Mandelbrot program for IBM PC in (D. Pountain), Sept 359
material selection program for construction in (T. Sawyer, M. Pecht), July 235
MetaWare Professional Pascal version 2.5, review (N. C. Shammas), Dec 265
Microsoft MS-Pascal version 3.31, review (N. C. Shammas), Dec 265
MIDI software for IBM PC in (D. Swearingen), June 211
Object Pascal object-oriented extension of, for Macintosh, Aug 177
Pascal Extender software for Macintosh (B. Webster), July 350
Pecan Software Systems UCSD Pascal version 4.2.1, review (N. C. Shammas), Dec 265
Personal Pascal software for Atari ST computers, July 347
poetry processing program in, Feb 224
polar method for generating normal deviates in (A. Latour), Aug 131
Prospero Software Pro Pascal version 2.14, review (N. C. Shammas), Dec 265
real-time in (J. Feldman), Sept 145
TOPSI 2.0 programming language in, review (L. Moskowitz), Aug 261
Turbo. *See* Turbo Pascal
windowing systems for Apple II in, Apr 101

Pattern-matching capabilities
of expert systems (P. W. Frey), Nov 161
on house architecture, Nov 166
on political events (P. A. Schrodt), Nov 177
of SNOBOL4 (J. F. Gimpel), Feb 175

PC6 computer from NCR, review (A. Little), Aug 241

PC-9801 computer from NEC, upgrades of (W. M. Raike), Jan 181

PC-9801VM2E personal computer from NEC, (W. M. Raike), Nov 347

PC-9801VM4 personal computer from NEC, (W. M. Raike), July 372

PC-DOS
DEBUG program and BREAKPT routine for (E. Batutis), Sept 127
PD PROLOG public domain implementation of Prolog for (R. Morein), Oct 155
Professional Debug Facility and Advanced Fullscreen Debug programs for, review (J. C. Carden), Apr 249

PCLAB laboratory interfacing software package, July 303

PC-Outline user-supported outliner (E. Shapiro), May 336

PC-Pedal input device, review (C. H. Pappas), May 285

PC-Shell (E. Shapiro), Oct 302

PCTEX typesetting package, review (H. R. Varian), Apr 267

PC Tools version 2.20, review (R. Rabinovitz), Oct 265

PC-Write word processor (E. Shapiro), Aug 320

PD PROLOG public domain implementation of Prolog (R. Morein), Oct 155

Pecan Software Systems
SCSD Pascal version 4.2.1, review (N. C. Shammas), Dec 265

Pellian equation (R. T. Kurosaka), May 379

Performance Index benchmark, IBM 158

Performance programming for IBM PC-compatible products (J. Rosenblum, D. Jacobs), IBM 181

Persoft Referee program (E. Shapiro), Nov 361

Personal Pascal software for Atari ST machines (B. Webster), July 347

Personal TEX PCTEX typesetting package, review (H. R. Varian), Apr 267

Personics SmartNotes program (E. Shapiro), Nov 362

Piano Player from Yamaha (W. M. Raike), Sept 351

Pibterm shareware communications program (E. Shapiro), Oct 300

PIE (Prolog Inference Engine) editor and compiler (S. Y. Blackwell), Oct 164

PIL Software Systems
ProComm shareware communications program (E. Shapiro), Oct 300

Pipelined architecture of Intel 80386 CPU (P. Wells), IBM 89

Plus Development
hardcard hard disk, review (E. White), May 273

Pocket APL language, review (E. H. Johnson), Mar 237

Poetry
machine reading of metric verse (P. Holzer), Feb 224
text processing of (M. Newman), Feb 221

Polar method for generating normal deviates (A. Latour), Aug 131

Political predictions with Holland bit-mapped classifier in Pascal (P. A. Schrodt), Nov 177

PopDrop program (E. Shapiro), Nov 362

Portable computers
Compaq Portable II model 3, review (S. Miastkowski), Oct 239
Fujitsu FM-16π (W. M. Raike), Jan 384, Mar 330
IBM PC Convertible, description of (G. M. Vose), IBM 83
Keystyle 80 keyboard and laptop computer (W. M. Raike), July 371
Panasonic Exec. Partner transportable computer, reviews (R. Malloy), Apr 231

PowerSoft Products
Super Utility version 1.10, review (R. Rabinovitz), Oct 265

Percept Instant Recall database manager (E. Shapiro), Oct 297

Predictions
on impact of nuclear detonations, with BASIC programs (J. R. Fanchi), Dec 143
political, with Holland bit-mapped classifier in Pascal (P. A. Schrodt), Nov 177

Printers
C. Itoh TriPrinter dot-matrix printer, review (R. D. Swearengin), Sept 283
Fujitsu DL-2400 20-pin dot-matrix, review (R. D. Swearengin), Nov 255
GE 3-8100 thermal dot-matrix, review (R. D. Swearengin), May 293
IBM Wheelprinter E, review (R. D. Swearengin), July 315
LaserWriter Plus, June 88
Lettrix resident print processor used with, review (A. R. Miller), May 299
Nisho NP-24110 24-pin dot-matrix, review (R. D. Swearengin), Nov 255
Printit printer interface card used with, Mar 261
Toshiba P321 24-pin dot-matrix, review (R. D. Swearengin), Nov 255

Printit printer interface card, review (H. Brugsch, J. J. Lazzaro), Mar 261

Probability analysis programs, on sucker bets (R. T. Kurosaka), Nov 373

Pro Code International
Waltz LISP version 5.01, review (W. Wong), July 293

ProComm shareware communications program (E. Shapiro), Oct 300

Prodigy 4 computer from Levco (B. Webster), Nov 323

ProDOS from Apple, catalog sort routine for (A. C. Silvestri), June 117

Product previews and descriptions
on Apple IIGS computer (G. Williams, R. Grehan), Oct 84
on Atari 520ST (J. R. Edwards, P. Robinson, B. McLaughlin), Jan 84
on Atari 1040ST computer (P. Robinson, J. R. Edwards), Mar 84
on Compaq Deskpro 386 computer (T. Thompson, D. Allen), Nov 84
on IBM PC Convertible (G. M. Vose), IBM 83
on LabView Laboratory Virtual Instrument Engineering Workbench (G. M. Vose, G. Williams), Sept 84
on Lotus Manuscript word processor (G. A. Stewart), Nov 91
on Q&A software (J. Edwards), Jan 120

Productivity Products
Objective-C object-oriented language
for Macintosh, Aug 183
programming experience with, Aug 202

Professional Debug Facility, review (J. C. Carden), Apr 249

Professional Pascal version 2.5, review (N. C. Shammas), Dec 265

Program listings
available from BYTE. *See* BYTE Listings
on Badfile utility in C language, Feb 158
on BREAKPT routine for PC-DOS DEBUG program, Sept 127
on Circuit Cellar data encryptor, Sept 109
on cyclic redundancy check calculations in C, Sept 119
on decision support system in Prolog, Nov 197
on diphantine equation solver in BASIC, Mar 346
on DOS functions from Turbo Pascal, Dec 104
on DRAGON program in Forth, Apr 138
on EditSort capsule editor and QuickSort capsule, Jan 148
on Euclid's algorithm, Jan 399
on file-indexing scheme, June 94
on finite-element analysis program for IBM PC, July 211
on frame-manipulation routines in Prolog, Jan 241
on free-form curves in BASIC, Dec 228

on Getspec assembly language routine for passing filenames to compiled BASIC, Nov 120
on graphing quadric surfaces with BASIC program for IBM PC, Dec 219
on Henon mapping in Turbo Pascal, Dec 163
on Hilbert curve program in BASIC, June 137
on interfacing machine language routine to Pascal, Sept 148
on interrupt routines for IBM PC in assembly language, IBM 252
on keyed file access in BASIC, Sept 140
on Mandelbrot program for Amiga in Lattice C, Dec 209
on memory manipulations for IBM PC, IBM 236
on MIDI software for IBM PC in Turbo Pascal, June 212
on musical fractals, June 188
on nuclear blast calculations, Dec 145
on number sequence programs in BASIC, Aug 334
on one-dimensional cellular automata program in BASIC, Dec 184
on Pan and Reif algorithm for inversion of large matrices, in BASIC, Apr 186
on Pellian equation solver, May 385
on Performance Index benchmark, IBM 160
on performance programming for IBM PC-compatible products, IBM 182
on ProDOS catalog sort routine, June 122
on SIMPL programs, Jan 134, Feb 111
on SmartWatch real-time clock in BASIC, Mar 120
on stress analysis of tunnels in underground mining, in BASIC, July 223
on Watson deductive reasoning program in Prolog, Nov 209
on windowing system, Apr 98

Programming
for anagram solving in Pascal (B. Keefer), July 113
for approximating integrals with Aitken extrapolation (D. M. Smith), Dec 113
of Badfile utility in C language for CP/M systems (L. Baker), Feb 157
of BREAKPT routine for PC-DOS DEBUG program (E. Batutis), Sept 127
in C language and assembly language, comparison of (T. Hogan), IBM 267
for calculation of cyclic redundancy checks (G. Morse), Sept 115
for CD-ROMs (B. Zoellick), May 177
of COLOR3D.BAS program for displaying molecules in color, for IBM PC (J. J. Farrell), Feb 149
for data compression with Huffman coding (J. Amsterdam), May 99
for dragon curve on Macintosh (B. R. Land), Apr 137
with Easy C preprocessor (P. Orlin, J. Heath), May 137
with Emulo-8 EPROM emulator (S. R. Ball), Apr 105
for extracting *n*th root from binary numbers (L. S. Wo), Nov 115
of file-indexing scheme (B. Webster), June 93
for flag testing and input comparison using 2^n property (R. C. Arp Jr.), Oct 145
functional, data flow in (C. Hankin, D. Till, H. Glaser), May 123
of Getspec assembly language routine for passing filenames to compiled BASIC (B. Hubanks), Nov 119
of Hilbert curve program in BASIC (M. Ackerman), June 137
for IBM PC-compatible products (J. Rosenblum, D. Jacobs), IBM 181
with Icon high-level programming language (R. E. Griswold, M. T. Griswold), Oct 167
of interrupt routines for IBM PC (W. J. Claff), IBM 249
with Let's C and csd C development package and symbolic debugger, review (W. G. Wong), Aug 267
in LISP language
with muLISP-86 development system, review (R. J. Schalkoff), Oct 249
with Visual Syntax editor (R. Levien), Feb 135
of Macintosh Explorer disassembler program (O. Andrade), Mar 145
mathematical laws in (C. A. R. Hoare), Aug 115
in Modula-2 (R. C. Corbeil, A. H. Anderson), May 111
with ITC Modula-2 Software Development System, review (M. Bridger), Oct 253
with reusable modules (N. C. Shammas), Jan 145
of spreadsheet program for Macintosh (J. Amsterdam), July 97
of storage allocators (J. Amsterdam), Oct 123
for Motorola MC 68000 processor, guidelines for (M. Morton), Sept 163
of MS-FORTRAN programs (M. Dahmke), IBM 217
object-oriented. *See* Object-oriented programming
of operating system functions from Turbo Pascal programs (D. F. Yriart), Dec 103
with OPS5 (Official Production System version 5) programming language (L. Moskowitz), Nov 217
and AUTO.OPS implementation of, Nov 222
and EsperOPS5 implementation for Macintosh (W. Jacobs), July 297
and TOPSI 2.0 implementation for IBM PC (L. Moskowitz), Aug 261
in PD PROLOG public domain implementation of Prolog (R. Morein), Oct 155
of polar method for generating normal deviates (A. Latour), Aug 131
rule-based (L. Moskowitz), Nov 217
with SIMPL programming language and compiler extensions of (J. Amsterdam), Feb 103
procedures and functions of (J. Amsterdam), Jan 131
of subroutine overlays in GWBASIC (M. Carmichael), May 151
of three-dimensional graphics program in BASIC, Jan 153
for displaying molecules in color (J. J. Farrell), Feb 149
of windows
basic principles in (B. Webster), Mar 129
implementation of (B. Webster), Apr 97
with write-once optical disks (J. R. Dulude), May 193

Prolog
from Arity
version 3.2, review (W. G. Wong), Mar 245
version 4.0, Mar 247
computer-vision system using, Jan 257
decision support system in (C. Y. Cuadrado, J. L. Cuadrado), Nov 193
Marvin program for teaching computers to learn in (A. T. Kolokouris), Nov 225
PD PROLOG public domain implementation of (R. Morein), Oct 155
PIE (Prolog Inference Engine) editor and compiler (S. Y. Blackwell), Oct 164
telecommunications in (L. Su), Oct 160
Turbo Prolog
implementation of (B. Webster), Sept 335
review (N. C. Shammas), Sept 293

Watson deductive reasoning program in (J.-C. Emond, A. Paulissen), Nov 207

Pro Pascal version 2.14, review (N. C. Shammas), Dec 265

Prospero Software Pro Pascal version 2.14, review (N. C. Shammas), Dec 265

Protected-mode program for IBM PC AT (R. P. Nelson), IBM 123

Pseudovariables in object-oriented programming, Aug 144

Public domain software
Abundance Forth-based database language (R. Green), Oct 193
for CP/M (B. N. Meeks), Oct 219
Enhanced Console Driver for user-friendly interface to DOS (A. Zackin), Oct 183
Icon high-level programming language (R. E. Griswold, M. T. Griswold), Oct 167
as issue theme (J. Edwards), Oct 153
PD PROLOG (R. Morein), Oct 155
Talking Moose desk accessory program for Macintosh (B. Webster), Dec 310
TEX digital typography system, Feb 169
Z80MU program for emulation of Zilog Z80 and CP/M 2.2 (R. A. Baumann), Oct 203

Publishing, TEX digital typography system in, Feb 169, 206

Puttkammer Software and Microcomputertechnich Advanced Fullscreen Debug, review (J. C. Carden), Apr 249

Q

Q&A software, preview (J. Edwards), Jan 120

QIC-60 tape backup unit, review (M. C. Rubel), Oct 243

Qmodem shareware communications program (E. Shapiro), Oct 300

Quadric surface (G. Haroney), Dec 215

Quantitative Micro Software microTSP version 4.1 program, review (P. Davenport), Apr 257

Queue simulation object-oriented program, Aug 230

Quicksoft PC-Write word processor (E. Shapiro), Aug 320

QuickSort capsule, Jan 148

QWERTY keyboard layout, efficiency of (D. W. Olson, L. E. Jasinski), Feb 241

R

Racal-Vadic modems, Dec 255
Racter program, review (H. Kenner), May 289

Raised Dot Computing
Braille-Edit talking word processor, Mar 199
review (H. Brugsch), Mar 251

RAM
DASCH external RAM disk for Macintosh (B. Webster), June 353
RAM-loadable character sets for IBM PC (R. Wilton), IBM 197

Ready! memory-resident outline processor from Living Videotext (E. Shapiro), Mar 297

Real-time systems
clocks in (S. Ciarcia), Mar 113
Hermes real-time multiprocessing operating system (R. C. Corbeil, A. H. Anderson), May 111
in Pascal (J. Feldman), Sept 145

Reed-Solomon for optical disk error correction (S. W. Golomb), May 203

Referee program (E. Shapiro), Nov 361

Reflex analytical database program, review (R. DeMaria), Aug 277

Relational database programs
filePro 16 and filePro 16 Harkness), Nov 297
Paradox 1.1, review (R. DeMaria), Sept 303

Reviews
of books. *See* Book reviews
of hardware. *See* Hardware reviews
of software. *See* Software reviews
of systems. *See* Systems reviews

Revolution Software Cruise Control program (E. Shapiro), Nov 361

RISC processor (D. Pountain), Jan 387
in IBM RT PC (R. S. Simpson), IBM 43

Robotics
artificial intelligence in (J. L. Cuadrado, C. Y. Cuadrado), Jan 237
autonomous navigation in (C. Jorgensen, W. Hamel, C. Weisbin), Jan 223
Camera systems in (P. Dunbar), Jan 161
in chemistry laboratory automation (G. W. Kramer, P. L. Fuchs), Jan 263
coordination of multiple manipulators in (J. S. Hawker, R. N. Nagel, R. Roberts, N. G. Odrey), Jan 203
increasing independence in aging (K. G. Engelhardt, R. Edwards), Mar 191
as issue theme (T. Clune), Jan 159
tactile sensors in (K. E. Pennywitt), Jan 172

Roland Corporation MPU-401 MIDI interface for IBM PC, June 211
review (R. Powell, R. Grehan), June 265

ROM
compact disk. *See* CD-ROM
EPROM programmers. *See* EPROM programmers
Kernel software of Amiga, introduction to (R. J. Mical), Feb 116

Round-off errors in computer approximations (S. L. Moshier), Apr 161

RT Personal Computer from IBM (R. O. Simpson), IBM 43

Rubicon Publishing SongPainter music software for Macintosh, review (M. S. Bernardo), June 273

Rule-base programming (L. Moskowitz), Nov 217

Runge-Kutta method for solving differential equations, Apr 216
in BASIC (D. M. Leo), Apr 196
in FORTRAN (B. Thomas), Apr 191

S

S-100 bus systems, best of BIX on, Oct 338

SALT laboratory interfacing software package, July 303

SB180 computer from Circuit Cellar
addition of SCSI to (S. Ciarcia), May 85, June 107
GT180 color graphics board for
basic technology of (S. Ciarcia), Nov 105
hardware of (S. Ciarcia), Dec 87

Science Accessories digitizers for data entry, review (E. D. Hearn), Nov 261

Scientific solutions LABPAC software, July 303

Scottsdale Systems Color Fox computer, review (J. D. Unger), Jan 301

SCSI bus, implementation of SB180 computer (S. Ciarcia), May 85, June 107

SDD.COM public domain file directory program, Oct 219

Security
of data, with Circuit Cellar data encryptor (S. Ciarcia), Sept 97
of home, overzealous system of (S. Ciarcia), Apr 85
of UNIX operating system (A. Filipske, Jan Hanko), Apr 113

Segmentation with Intel 80386 CPU, IBM 89, 111

Semantic compaction system, in speech impairment (B. R. Baker), Mar 160

practical applications of, Mar 166

Sensors in robotics
in automation of chemistry laboratory, Jan 274
in navigation (C. Jorgensen, W. Hamel, C Weisbin), Jan 223
tactile (K. E. Pennywitt), Jan 172
visual (P. Dunbar), Jan 161

Serial EPROM programmer, intelligent, construction of (S. Ciarcia), Oct 103

Shareware (E. Shapiro), Oct 297

Shivji, Shiraz, interview with (P. Robinson), Mar 90

Shows and exhibits. *See* Exhibits and shows

Sider hard disk drive, review (D. E. Hall), Jan 319

Sieve of Eratosthenes benchmarks. *See* Benchmarks

Signal Technology ILS laboratory interfacing software, July 303

Silicon Beach Software Dark Castle game for Macintosh (E. Shapiro), Dec 327

Similarity mapping program for Macintosh in BASIC (R. Spencer), Aug 85

SIMPL programming language and compiler extensions of (J. Amsterdam), Feb 103
procedures and functions of (J. Amsterdam), Jan 131

SimpleSoft Products It Figures spreadsheet program (E. Shapiro), July 381

Simulation
with CHEMMOD molecular modeling system (D. Pountain), Dec 334
of circuits, program for
for Commodore 64 (D. McNeill), July 170
introduction to (W. Blume), July 165
SPICE (Simulation Program for Integrated Circuit Engineering), July 165, 170
of customers queuing in bank, object-oriented program on, Aug 230
of financial investments with Blue Chip software (E. Shapiro), Dec 326
with STELLA, May 335, Dec 277
of surgery with Surgeon program for Macintosh (E. Shapiro), Dec 322

Singular Software Interlace Database program (E. Shapiro), May 336

S&K Technology Strike version spelling checker, review (R. Ramsey), Nov 289

SKAM routines for keyed file access in BASIC (S. C. Perry), Sept 137

Small Computer Company filePro 16 and filePro 16 Plus database management software, review (R. Harkness), Nov 297

Smalltalk language
Animal Game in, Aug 153
improving efficiency of (C. B. Duff), Aug 211
for Macintosh, Aug 177
Methods version of, programming experience with, Aug 206
syntax and design philosophy of (T. Kaehler, D. Patterson), Aug 145
Tower of Hanoi program in, Aug 146

SmarTime real-time system, Mar 123

Smartmodems from Hayes, June 339, Dec 255

SmartNotes program (E. Shapiro), Nov 362

Smart QIC-File tape backup unit, review (M. C. Rubel), Oct 243

SmartWatch real-time clock from Dallas Semiconductor, Mar 118

SNOBOL4 language, processing strings in (J. F. Gimpel), Feb 175

SoftKlone Distributing Software Carousel virtual memory manager, review (M. Haas), Sept 299

Software
for abstract mathematical art, with BASIC program for IBM PC (K. E. Perry), Dec 181
Acta outline processor for Macintosh (B. Webster), Dec 316
AI:Typist word processor (E. Shapiro), July 383
for Amiga (B. Webster), Apr 322
Amiga ROM Kernel, introduction to (R. J. Mical), Feb 116
for animation of Amiga graphics, Sept 241, Nov 285
for Atari 520ST (B. Webster), Apr 318
Balance of Power game (E. Shapiro), Mar 299, Dec 322
Blue Chip financial investment simulation programs (E. Shapiro), Dec 326
Boxcalc spreadsheet program (E. Shapiro), May 336
Boxes and Arrows spreadsheet and graphics program (E. Shapiro), May 335
BYTE listings of. *See* BYTE listings
for CD-ROM, development of (B. Zoellick), May 177
CGEN program for translating BASIC into C (D. Pountain), Oct 311
CHEMMOD molecular modeling system (D. Pountain), Dec 334
circuit simulation programs
for Commodore 64 (D. McNeill), July 170
introduction to (W. Blume), July 165
SPICE (Simulation Program for Integrated Circuit Engineering), July 165, 170
ComicWorks painting program for Macintosh (E. Shapiro), Dec 321
for CP/M in public domain (B. N. Meeks), Oct 219
Cruise Control cursor control program (E. Shapiro), Nov 361
Dac-Easy Word word processor (E. Shapiro), July 381
Datatext word processor (E. Shapiro), July 383
dBASE III Plus (E. Shapiro), June 330
debugging of. *See* Debugging
for Definicon 68020 coprocessor (T. Marshall, C. Jones, S. Kluger), Aug 108
Easy word processor, Mar 300
Enhanced Console Driver for user-friendly interface to DOS (A. Zackin), Oct 183
finite-element analysis program for IBM PC (R. W. Johnson, F. G. Loygorri), July 199
Framework II (E. Shapiro), Apr 331
FullPaint program for Macintosh (E. Shapiro), Sept 368
Henon mapping program in Turbo Pascal (G. Hughes), Dec 161
IBM PC-compatible, MacCharlie accessory for Macintosh providing access to (L. Crockett), Feb 262
on impact of nuclear detonations in BASIC (J. R. Fanchi), Dec 143
Induce program for extracting knowledge from data, in Turbo Pascal for IBM PC, Nov 158
Instant Music program for Amiga (B. Webster), Dec 308
Instant Recall database manager (E. Shapiro), Oct 297
Interlace database program (E. Shapiro), May 338
Intuitive Solution business application generator (D. Pountain), May 363
It Figures spreadsheet program (E. Shapiro), July 381
KeepTrack Plus (E. Shapiro), June 334
for keyed file access in BASIC (S. C. Perry), Sept 137
Lattice Screen Editor program editor for Amiga, July 352
of Lotus/Intel/Microsoft Expanded Memory Specification (R. Duncan), IBM 169
for Macintosh (B.

Webster), Apr 315
MacView program for transferring Macintosh graphics to IBM PC (M. Anacker), June 131
Mandelbrot program
for Amiga in Lattice C (P. B. Schroeder), Dec 207
for IBM PC in Turbo Pascal (D. Pountain), Sept 359
Manuscript word processor, preview (G. A. Stewart), Nov 91
Mapper similarity mapping program for Macintosh, in BASIC (R. Spencer), Aug 85
Marvin program in Prolog for teaching computers to learn (A. T. Kolokouris), Nov 225
for material selection in construction
with MSP program in Turbo Pascal (T. Sawyer, M. Pecht), July 235
with physical property estimation programs in BASIC (J. N. Stone), July 253
Metascope debugger for Amiga (B. Webster), Nov 334
Mindreader word processor (E. Shapiro), Aug 319
Mirror telecommunications program (E. Shapiro), Apr 329
More outline processor for Macintosh, Sept 367, Dec 316
Music
in C language, for IBM PC in MIDI interface (J. Kubicky), June 199
ConcertWare+ and SongPainter programs, review (M. S. Bernardo), June 273
for Macintosh, June 171, 273
overview of (R. Powell), June 145
in Turbo Pascal, for IBM PC and MIDI interface (D. Swearingen), June 211
Orpheus poetry processing program, Feb 221
Pascal Extender for Macintosh, July 350
PC-Outline user-supported outliner (E. Shapiro), May 336
PC-Shell (E. Shapiro), Oct 302
PC-Write word processor (E. Shapiro), Aug 320
PD PROLOG implementation of Prolog (R. Morein), Oct 155
Personal Pascal for Atari ST machines, July 347
Pibterm shareware communications program (E. Shapiro), Oct 300
PopDrop program (E. Shapiro), Nov 362
ProComm shareware communications program (E. Shapiro), Oct 300
ProDOS catalog sort routine (A. C. Silvestri), June 117
program listings of. *See* Programming
protected-mode program for IBM PC AT (R. P. Nelson), IBM 123
in public domain. *See* Public domain software
Q&A package, preview (J. Edwards), Jan 120
Qmodem shareware communications program (E. Shapiro), Oct 300
for RAM-loadable character sets for IBM PC (R. Wilton), IBM 198
Ready! memory-resident outline processor, Mar 297
Referee program (E. Shapiro), Nov 361
reviews. *See* Software reviews
SmartNotes program (E. Shapiro), Nov 362
Stella program, May 335, Dec 277
for stress analysis of tunnels in underground mining (D. L. Petersen, S. L. Crouch), July 219
Surgeon surgery simulation program for Macintosh (E. Shapiro), Dec 322
Talking Moose desk accessory program for Macintosh (B. Webster), Dec 310
for text analysis and identification of unknown authors (J. Tankard), Feb 231
Truss2 bridge-truss analysis program in BASIC (C. Pedicini), July 145
TSRCOM utilities (E. Shapiro), Oct 299
Turbo Editor Toolbox, Mar 297
TxEd program editor for Amiga (B. Webster), Nov 336
Visual syntax editor for visual programming in LISP (R. Levien), Feb 135
Volkswriter 3 (E. Shapiro), June 332
Watson deductive reasoning program in Prolog (J.-C. Emond, A. Paulissen), Nov 207
Word 3.0 word processor, Aug 322, Oct 261
WordStar 2000 Release 2 (E. Shapiro), June 329
Workbench program for object-oriented programming, Aug 162
Works integrated package for Macintosh (E. Shapiro), Nov 365
Z80MU program for emulation of Zilog Z80 and CP/M 2.2 (R. A. Baumann), Oct 203

Software Group Enable integrated software package
version 1.0, review (S. King), Jan 331
version 1.1 (R. Malloy), Jan 334

Software Carousel virtual memory manager, review (M. Haas), Sept 299

Software ICs, reusable, in object-oriented programming (B. Cox, B. Hunt), Aug 161

Software reviews
on Advanced Fullscreen Debug (J. C. Carden), Apr 249
on Aegis Animator and Images animation and painting programs (W. Block), Nov 285
on Arity/Prolog for MS-DOS systems (W. G. Wong), Mar 245
on Braille-edit (H. Brugsch), Mar 251
on BYSO LISP version 1.17 (W. Wong), July 293
on DeSmet C Development Package for Macintosh (J. Robie), Aug 253
on Eco-C88 C compiler (D. D. Clark), Jan 307
on Enable integrated package (S. King), Jan 331
on ExperOPS5 programming language for Macintosh (W. Jacobs), July 297
on filePro 16 and filePro 16 Plus database management programs (R. Harkness), Nov 297
on flash-Com electronic mail and telecommunications system (B. N. Meeks), Dec 281
on GEM Draw painting program (R. Birmele), May 269
on Guru integrated package (E. R. Tello), Aug 281
on KAMAS outline processor (A. S. Woodhull), Apr 241
on laboratory interfacing packages (P. Wirth, L. E. Ford), July 303
on Lattice C compilers for Amiga (C. Heath), Nov 271
for IBM PC version 2.15 (D. S. Woolston), Feb 273
on Let's C and csd development package and symbolic debugger (W. G. Wong), Aug 267
on Lettrix resident print processor (A. R. Miller), May 299
on MacDraw painting program (R. Birmele), May 269
on Manx Aztex C68K C compiler for Amiga (C. Heath), Nov 271
on Microsoft Word version 3.0 word processor (M. C. Rubel), Oct 261
on MicroTEX typesetting package (H. R. Varian), Apr 267
on microTSP version 4.1 time-series regression package (P. Davenport), Apr 257
on Mix C compiler for IBM PC (R. Grehan), June 257
on Modula-2 Software Development system (M. Bridger), Oct 255
on Modula-2 System for

Z80 CP/M (B. R. Anderson), Mar 225
on muLISP-86 LISP development system (R. J. Schalkoff), Oct 249
on NewWord 3 word processor (J. Heilborn, N. Reel), Aug 273
on Norton Utilities version 3.1 (R. Rabinovitz), Oct 265
on Paradox 1.1 relational database (R. DeMaria), Sept 303
on Pascal packages for IBM PC (N. C. Shammas), Dec 265
on PC Tools version 1.10 (R. Rabinovitz), Oct 265
on PCTEX typesetting package (J. R. Varian), Apr 267
on Pocket APL language (E. H. Johnson), Mar 237
on Professional Debug Facility (J. C. Carden), Apr 249
on Racter program (H. Kenner), May 289
on Reflex analytical database (R. DeMaria), Aug 277
on Software Carousel virtual memory manager (M. Haas), Sept 299
on SPSS/PC+ analytical package (J. M. Jacques), Nov 270
on STELLA modeling and simulation program for Macintosh (S. B. Robinson), Dec 277
on Strike spelling checker (R. Ramsey), Nov 289
on Super Utility version 1.10 (R. Rabinovitz), Oct 265
on TOPSI 2.0 programming language for IBM PC (L. Moskowitz), Aug 261
on Turbo Lightning spelling checker (R. Ramsey), Sept 293
on Turbo Pascal version 3.0 (M. Bridger), Feb 281
on Waltz LISP Version 5.01 (W. Wong), July 293
on WordPerfect 4.1 word processor (R. Birmele), Sept 311
on ZBasic interactive BASIC compiler (T. J. Byers), May 265

Software Show of 1985 in Japan, highlights of (W. M. Raike), Feb 317

Soft Warehouse muLISP-86 programming language, review (R. J. Schalkoff), Oct 249

SoftWorks Development PC-Outline user-supported outliner (E. Shapiro), May 336

Sonar sensors in robotic navigation, Jan 230

SongPainter music software for Macintosh, review (M. S. Bernardo), June 273

Sound system of Amiga (D. D. Thiel), Oct 139

Soviet Union, microelectronic technology in (P. Walton), Nov 137

Speech
Braille-Edit talking word processor, Mar 199, 251
Kurzweil Voice Writer voice-activated word processor (R. Kurzweil), Mar 177
semantic compaction system for speech-impaired (C. R. Baker), Mar 160
speech synthesizers for blind users, Mar 199, 251, 261
Talking Moose public domain desk accessory for Macintosh (B. Webster), Dec 310
voice synthesis on Amiga (D. D. Thiel), Oct 140

Spelling checkers
Strike version 1, review (R. Ramsey), Nov 289
Turbo Lightning version 1.00A, review (R. Ramsey), Nov 289

Sperry PC/IT computer, review (F. D. Davis), Aug 247

SPICE (Simulation Program for Integrated Circuit Engineering), July 165
for Commodore 64 (D. McNeill), July 170

Spreadsheets
with Boxcalc program, May 336
with Boxes and Arrows program, May 335
with Enable integrated software package, Jan 331
with It Figures program, July 381
in Modula-2, programming project on (J. Amsterdam), July 97

SPSS/PC+ analytical software package, review (J. M. Jacques), Nov 279

Standards
of ANSI on C language (S. A. Hersee, D. Knopoff), Mar 135
on musical instrument digital interface. *See* MIDI specifications

STELLA program (E. Shapiro), May 335
review (S. B. Robinson), Dec 277

Storage allocation in Modula-2, programming project on (J. Amsterdam), Oct 123

Street Electronics Echo speech synthesizers, Mar 199, 251, 261

Stress analysis of tunnels in underground mining (D. L. Petersen, S. L. Crouch), July 219

Strike version 1 spelling checker, review (R. Ramsey), Nov 289

String processing in SNOBOL4 (J. F. Gimpel), Feb 175

Structural analysis in engineering
with finite-element analysis method, July 145, 148, 199
on IBM PC (R. W. Johnson, F. G. Loygorri), July 199
with Truss2 bridge-truss analysis program (C. Pedicini), July 145

STSC Pocket APL language, review (E. H. Johnson), Mar 237

Subroutine overlays in GWBASIC (M. Carmichael), May 151

Summagraphics digitizers for data entry, review (W. D. Hearn), Nov 261

Super Utility version 1.10, review (R. Rabinovitz), Oct 265

Surgeon surgery simulation program for Macintosh (E. Shapiro), Dec 322

Symantec Corporation Q&A software package, preview (J. Edwards), Jan 120

Symmetry Corporation Act outline processor for Macintosh (B. Webster), Dec 316

Synchronization of processes Modula-2, May 111

Syntech MIDI interfaces for Apple II and Commodore 64, review (R. Powell, R. Grehan), Jan 265

Synthesizers
digital music
Kurzweil 250 Digital Synthesizer, review (C. Morgan), June 279
for Macintosh (C. Yavelow), June 171
overview of (R. A. Moog), June 155
speech, for blind users, Mar 199, 251, 261

Sysgen Smart QIC-File tape backup unit, review (M. C. Rubel), Oct 243

System reviews
on Atari 520ST computer (E. Jensen), June 233
on AT&T UNIX PC (A. J. W. Mayer), May 254
on Canon A-200 computer system (P. V. Callamaras), Jan 293
on Commodore Amiga 1000 (T. Thompson), Oct 231
on Compaq Deskpro 286 computer Models 1 and 2 (S. Miastkowski), June 243
on Compaq Portable II computer Model 3 (S. Miastkowski), Oct 239
on Conquest Turbo PC computer (J. D. Unger), July 289
on Epson Equity I computer (J. D. Unger), Nov 239
on Epson Equity III computer (W. Rash Jr.), Dec 239
on Franklin ACE 2200 computer (A. S. Woodhull), Sept 263
on ITT XTRA XP computer (J. D. Unger), July 281
on Kaypro 286i computer (H. Krause), Mar 217
on Kaypro PC computer (J. D. Unger), Nov 239

on Leading Edge Model D PC computer (S. Miastkowski), Sept 269
on MacCharlie (L. Crockett), Feb 262
on Macintosh Plus computer (C. Crawford), Nov 247
on Motorola VME/10 (R. R. Robinson III), Feb 253
on Multitech MPF-PC/700 D1 computer (J. D. Unger), Nov 239
on NCR PC6 computer (A. Little), Aug 241
on Panasonic Exec. Partner computer (R. Malloy), Apr 231
on Scottsdale Systems Color Fox computer (J. D. Unger), Jan 301
on Sperry PC/IT computer (F. D. Davis), Aug 247
on TeleVideo Tele-286 Model 2 computer (W. Rash Jr.), June 251
on Western AT computer (W. Rash Jr.), Dec 239
on Xerox 6060 computer (W. Rash Jr.), Sept 275
on Zenith Z-241 computer (W. Rash Jr.), Dec 239
on Zenith Z-248 computer (W. Rash Jr.), Dec 239

Systems programming with Modula-2 (R. C. Corbeil, A. H. Anderson), May 111

T

Tactile sensing in robotics (K. E. Pennywitt), Jan 172

Taiwan Computex show of 1986 (W. M. Raike), Oct 307

Talking Moose public domain desk accessory for Macintosh (B. Webster), Dec 310

Talking word processor for visually impaired, Braille-Edit, Mar 199, 251

Tallgrass Technologies TC-4060 tape backup unit, review (M. C. Rubel), Oct 243

Tape backup systems
comparison of (A. Antonuccio), May 227
hardware reviews (M. C. Rubel), Oct 243

TDI Software Modula-2/ST compiler (B. Webster), Feb 332

TDS-AP MIDI interface for Apple II computers, review (R. Powell, R. Grehan), June 265

Technical writing (C. Weston), Nov 94
Lotus Manuscript word processor for, preview (G. A. Stewart), Nov 91
with U-MAN 1000 computer (D. Pountain), Dec 329

Tecmar
Lab Master board, software interfacing packages for (P. Wirth, L. E. Ford), July 303
QIC-60 tape backup unit, review (M. C. Rubel), Oct 243

Tele-286 Model 2 computer from TeleVideo Systems, review (W. Rash Jr.), June 251

Telecommunications. *See* Communications

Telecommuting (J. M. Tazelaar), Mar 155

TeleLearning, Electronic University Network of (D. Osgood), Mar 171

Telesensor Systems Versabraille, Mar 199

TeleVideo Systems Tele-286 Model 2 computer, review (W. Rash Jr.), June 251

TEX typesetting systems, Feb 206
comparison of PCTEX and MicroTEX implementations of (H. R. Varian), Apr 267
Knuth interview on (G. M. Vose, G. Williams), Feb 169

Texas Instruments
Professional Computer, BASIC program for free-form curves on (S. Enns), Dec 225
TMS34010 Graphics System Processor (C. R. Killebrew Jr.), Dec 193

TEXT GOBBLER programs for identifying unknown authors, Feb 232

Text Processing
in Arabic and other problem scripts (P. A. MacKay), Feb 201
as computer science problem, Knuth interview on (G. M. Vose, G. Williams), Feb 169
as issue theme (J. R. Edwards), Feb 167
keyboard efficiency in (D. W. Olson, L. E. Jasinski), Feb 241
in natural language (J. Pollack, D. L. Waltz), Feb 189
of poetry (M. Newman), Feb 221
programs for identifying unknown authors (J. Tankard), Feb 231
in SNOBOL4 language (J. F. Gimpel), Feb 175
TEX typesetting system in, Feb 169, 206
comparison of PCTEX and MicroTEX implementations of (H. R. Varian), Apr 267
Knuth interview on (G. M. Vose, G. Williams), Feb 169

TG-4060 tape backup unit, review (M. C. Rubel), Oct 243

Think Technologies Lightspeed C development environment for Macintosh (B. Webster), Aug 323, Sept 340

Three-dimensional graphics, BASIC program for (H. Mittelbach), Jan 153
for molecules in color (J. J. Farrell), Feb 149
on Quadric surfaces (G. Haroney), Dec 215

Time-series analysis, Apr 262
with microTSP program version 4.1, review (P. Davenport), Apr 257

Titanic, image-processing technology used in locating (M. Spalding, B. Dawson), Mar 97

TML MacLanguage Series Pascal compiler (B. Webster), Feb 340, Apr 315

TMS34010 Graphics System Processor from Texas Instruments (C. R. Killebrew Jr.), Dec 193

TOPSI 2.0 programming language for IBM PC, review (L. Moskowitz), Aug 261

TOS operating system of Atari ST computer, and Motorola MC68000 processor (M. Rothman), Sept 223

Toshiba P321 24-pin dot-matrix printer, review (R. D. Swearengin), Nov 255

Touch sensing, robotic (K. E. Pennywitt), Jan 172

Tower of Hanoi problem, Smalltalk program on, Aug 146

Transputer from Inmos (D. Pountain), July 363
Meiko Computing Surface based on (D. Pountain), July 364

Tri-Data OZ Guardian Model 533 modem, Dec 255

Tripos operating system from Metacomco (D. Pountain), Feb 321

TriPrinter, C. Itoh Model 20, review (R. D. Swearengin), Sept 283

TRS-80, physical property estimation programs in BASIC for (J. N. Stone), July 253

Truss2 bridge-truss analysis program in BASIC (C. Pedicini), July 145

TRW vector processor in micro-based supercomputer (N. H. Christ, A. E. Terrano), Apr 145

TRSCOM utilities (E. Shapiro), Oct 299

Turbo Editor Toolbox from Borland (E. Shapiro), Mar 297

Turbo Lightning version 1.00A spelling checker, review (R. Ramsey), Nov 289

Turbo Pascal
anagram solving in (B. Keefer), July 113
compared to ITC Modula-2 (M. Bridger), Oct 255

file-indexing program in, June 96
Henon mapping in (G. Hughes), Dec 161
Induce program for extracting knowledge from data in,
for IBM PC, Nov 158
for Macintosh (B. Webster), Nov 338
MacView program for transferring Macintosh graphics to IBM PC in, June 131
Mandelbrot program for IBM PC in (D. Pountain), Sept 359
material selection program for construction in (T. Sawyer, M. Pecht), July 235
MIDI software for IBM PC in (D. Swearingen), June 211
TOPSI 2.0 programming language in, review (L. Moskowitz), Aug 261
TurboPower Programmer's Utilities in, Feb 334
using operating system functions from programs written in (D. F. Yriart), Dec 103
version 3.0, review (M. Bridger), Feb 281

TurboPower Software
Programmer's Utilities (B. Webster), Feb 334
TRSCOM utilities (E. Shapiro), Oct 299

Turbo Prolog
programming language (B. Webster), Sept 335
review (N. C. Shammas), Sept 293

Turner Hall Card for IBM PC memory expansion, review (J. Angel), Sept 287

TxEd program editor for Amiga (B. Webster), Nov 336

Typesetting
in Arabic and other problem scripts (P. A. MacKay), Feb 201
METAFONT system in, Feb 169, 211
TEX digital system in, Feb 169, 206
comparison of PCTEX and MicroTEX packages (H. R. Varian), Apr 267
Knuth interview on (G. M. Vose, G. Williams), Feb 169

U

UCSD Pascal version 4.2.1. from Pecan Software systems, review (N. C. Shammas), Dec 265

U.K., BYTE reports from (D. Pountain)
on Acorn RISC machine, Jan 387
on AmigaDOS, Feb 321
on Amstrad PCW 8256 computer and word processor, Mar 333
on Apricot XEN computer, Apr 305
on CGEN program for translating BASIC into C, Oct 311
on CHEMMOD molecular model system, Dec 334
on Inmos Transputer, July 363
on Intuitive Solution business application generator, May 363
on Mandelbrot program for IBM PC in Turbo Pascal, Sept 359
on Mello Computing Surface, July 364
on Miracle Technology WS3000 modem, June 319
on Occam language, July 363, Sept 359
and dynamic load balancing, July 368, Sept 359
on U-MAN 1000 computer, Dec 329
on wafer-scale integration of semiconductor devices, Nov 351

U-MAN 1000 computer (D. Pountain), Dec 329

Underground tunnels in mining, stress analysis of (D. L. Petersen, S. L. Crouch), July 219

Underwater exploration with image processing technology (M. Spalding, B. Dawson), Mar 97

Universal Data Systems FasTalk 1200 modem, Dec 255

UNIX
compatibility with Motorola MC68000 architecture (A. L. Rood, R. C. Cline, J. A. Brewster), Sept 179
security features of (A. Filipski, J. Hanko), Apr 113
System V version 2 in AT&T UNIX PC, May 254

UNIX PC from AT&T, review (A. J. W. Mayer), May 254

User interface
with Amiga ROM Kernel and Intuition, Feb 124
in hearing impairment, Mar 177
iconic, in object-oriented programming (B. Cox, B. Hunt), Aug 161
performance of (J. Uebbing, C. Young), Aug 176
to operating system with Enhanced Console Driver (A. Zackin), Oct 183
in visual impairment, Mar 199, 251

USRobotics Courier 2400 modem, Dec 255

V

Vectra-D Dual-Mode Workstation (W. M. Raike), Sept 351

Ven-Tel 2400 Plus Modem, Dec 255

Versabraille from Telesensory Systems, Mar 199

Versatron Footmouse input device, review (C. H. Pappas), May 285

Video
construction of audio-and-video multiplexer (S. Ciarcia), Feb 85
glossary on, Jan 168

Videotrax tape backup unit, review (M. C. Rubel), Oct 243

Virtual machines
(J. Shiel), IBM 111
LabView Laboratory Virtual Instrument Engineering Workbench, preview (G. M. Vose, G. Williams), Sept 84
VM2, SIMPL compiler for (J. Amsterdam), Jan 131, Feb 103

Virtual memory (J. Shiell), IBM 111
with Intel 80386 CPU, IBM 111
Software Carousel virtual memory manager, review (M. Haas), Sept 299

Vision in robotics
artificial intelligence in (J. L. Cuadrado, C. Y. Cuadrado), Jan 237
camera systems in (P. Dunbar), Jan 161

Visual programming in LISP, with Visual Syntax editor (R. Levien), Feb 135

Visual Syntax editor, for visual programming in LISP (R. Levien), Feb 135

Visually impaired persons
Braille-Edit talking word processor for, Mar 199
review (H. Brugsch), Mar 251
workable computing system for (A. Arditi, A. E. Gillman), Mar 199

VM2 virtual machine, SIMPL compiler for (J. Amsterdam), Jan 131, Feb 103

VME/10 system from Motorola, review (R. E. Robinson III), Feb 253

Voice. *See* Speech

Volkswriter 3 software (E. Shapiro), June 332

Votrax Personal Speech System, Mar 199, 261

W

Wafer-scale integration of semiconductor devices (D. Pountain), Nov 351

Waltz LISP version 5.01, review (W. Wong), July 293

Warp Speed Light Pen input device, review (C. H. Pappas), May 285

Watson deductive reasoning program in Prolog (J.-C. Emond, A. Paulissen), Nov 207

West Coast Computer Faire of 1986 (B. Webster), Aug 323

Western AT computer, review (W. Rash Jr.), Dec 239

Western Automation Laboratories DASCH external RAM disk for Macintosh, June 353

Western Design Center W65C816 processor, in Apple IIGS computer, Oct 84

Wheelprinter E from IBM, review (R. D. Swearengin), July 315

Windows
of AT&T UNIX PC, May 254
with Enable integrated software package, Jan 331
Microsoft, of Apricot XEN computer, Apr 309
programming project on basic principles in (B. Webster), Mar 129
implementation of (B. Webster), Apr 97

Wisper wafer from Anamartic (D. Pountain), Nov 351

Word 3.0 word processor (E. Shapiro), Aug 322
review (M. C. Rubel), Oct 261

WordPerfect 4.1 word processor, review (R. Birmele), Sept 311

Word processing
with AI:Typist software (E. Shapiro), July 383
with Anstrad PCW 8256 computer (D. Pountain), Mar 333
with Braille-Edit, Mar 199
review (H. Brugsch), Mar 251
with Dac-Easy Word program (E. Shapiro), July 381
with Datatext software (E. Shapiro), July 383
with Easy program, Mar 300
with Enable integrated software package, Jan 331
in Japanese language on IBM PC (W. M. Raike), Aug 330
with Kurzweil Voice Writer (R. Kurzweil), Mar 177
with Lotus Manuscript program, preview (G. A. Stewart), Nov 91
with Mindreader program (E. Shapiro), Aug 319
with NewWord 3 program, review (J. Heilborn, N. Reel), Aug 273
with PC-Write program (E. Shapiro), Aug 320
and processing text as issue theme (J. R. Edwards), Feb 167.
See also Text processing
with Q&A software package, Jan 120
with Word 3.0 program (E. Shapiro), Aug 322
review (M. C. Rubel), Oct 261
with WordPerfect 4.1 program, review (R. Birmele), Sept 311

WordStar 2000 Release 2 software (E. Shapiro), June 329

Workbench program for object-oriented programming, Aug 162

Works integrated package for Macintosh (E. Shapiro), Nov 365

Write-once optical disks, May 215
programming with (J. R. Dulude), May 193

WS3000 modem from Miracle Technology (D. Pountain), June 319

X

XBIOS of Atari ST computers, Sept 223

XEN computer from Apricot (D. Pountain), Apr 305

Xerox 6060 computer, review (W. Rash Jr.), Sept 275

XLISP, EDITNET program in, Feb 196

XOR function, in calculation of cyclic redundancy checks (G. Morse), Sept 115

Y

Yamaha
CX5-M music computer, musical fractal program for, June 185, 196
Piano Player (W. M. Raike), Sept 351

Yokogawa Hewlett-Packard Vectra-D Dual-Mode Workstation (W. M. Raike), Sept 351

Z

Z80-based CP/M systems
Amstrad PCW 8256 computer and word processor, Mar 333
Commodore 128 personal computer, review (W. Wiese Jr.), July 269
KAMAS outline processor for, Apr 241
Modula-2 System for, review (B. R. Anderson), Mar 225
Z80MU program for emulation of Zilog Z80 and CP/M 2.2 (R. A. Baumann), Oct 203

Z80MU program for emulation of Zilog Z80 and CP/M 2.2 (R. A. Baumann), Oct 203

Z-241 computer from Zenith, review (W. Rash Jr.), Dec 239

Z-248 computer from Zenith, review (W. Rash Jr.), Dec 239

ZBasic interactive BASIC compiler, review (T. J. Byers), May 265

Zenith
Z-248 computer, review (W. Rash Jr.), Dec 239
Z-241 computer, review (W. Rash Jr.), Dec 239

Zilog Z80 processor and CP/M 2.2, Z80MU program for emulation of (R. A. Baumann), Oct 203

Zymark Zymate Laboratory Robot, Jan 263

Zymate Laboratory Robot from Zymark, Jan 263

BBS'S POSTING BYTENET LISTINGS

Australia:

Grayham Smith
12 Brentwood Road
Flinders Park, South Australia 5025
The Electronic Oracle
300 Baud, CCITT Standard
Telephone: 08-43-3331 Voice
08-260-6686 BBS

Edward A. Romer
31 Warwick Street
Killara,
Sydney NSW, Australia 2071
OMEN
300 & 1200 Baud
Telephone: 02-498-2399
Voice (Work)
02-499-2642 Voice (Home)
02-498-2495 BBS

Alan Salmon
PCUG Sysop
GPO Box 2229
Canberra,
A.C.T. 2601, Australia
Canberra PC Users Group Inc.
300 & 1200 Baud
Telephone: 61-62-58-9967 BBS

Angus S. Bliss
POB 293
Hamilton NSW 2303, Australia
Newcastle Microcomputer Club
300 Baud, CCITT Standard, 8 Bits, 1 Stop, No Parity
Telephone: 049-67-2433 Voice (Angus Bliss)
049-54-9505 Voice (Tony Nicholson)
61-49-685385 BBS

John Hastwell-Batten
POB 242
Dural, NSW 2158, Australia
Tesseract RCPM+
300 Baud, CCITT Standard, 8 Bits, No Parity
Telephone: 02-651-2363 Voice
02-651-1404 BBS

Phil Harding
POB 35
Charnwood A.C.T., Australia 2615
PC-Exchange Bulletin Board
300 & 1200 Baud, CCITT Standard
Telephone: 61-062-581406 Voice
61-62-586352 BBS

Eric Salter
POB 60
Canterbury 3126, Australia
MICOM: The Microcomputer Club of Melbourne
300 Baud
Telephone: 61-3-861-9117 Eric Salter
61-3-762-1386 Peter Jetson (SYSOP)
61-3-762-5088 BBS

Craig Bowen
29 Warrigal Road
Surrey Hills 3127, Vic., Australia
Public Resource #1
300 Baud, CCITT Standard, 8 Bits, 1 Stop, No Parity
Telephone: 03-890-2174

John Blackett-Smith
Unit 8
69 Wattle Road
Hawthorn 3122, Australia
The National Fido
Telephone: 613-818-2336

Austria:

Wolfgang Hryzak
Bahnstrasse 48
A-2230 Ganserndorf, Austria
University of Vienna BBS FIDO
300 Baud, 8 Bits, 1 Stop Bit
Telephone: 02282-24094 BBS

Brazil:

Sistema Sampa
ATTN: Rizieri Maglio
R. Portugal, 202
Jdm Europe - CEP 01446
Sao Paulo - SP - Brazil
Sistema Sampa
300 & 1200 Baud, CCITT Standard
Telephone: 011-8536273 BBS

Canada:

Leigh Calnek
3036 25th Avenue
Regina, Saskatchewan, Canada S4S 1K9
Telephone: 306-586-9253 BBS

Tom Kashuba
PCOMM Systems
1411 Fort Street, Suite 2001
Montreal, Quebec, Canada H3H 2N7
Telephone: 514-989-9450 BBS

Gary McCallum
Western Canadian Distribution Center
3420 48th Street
Edmonton, Alberta, Canada T6L 3R5
300 & 1200 & 2400 Baud
Telephone: 403-462-9189 Voice
403-461-9124 BBS

Judson Newell
Canada Remote Systems
Suite 311, 4198 Dundas Street West
Toronto, Ontario, Canada M8X 1Y6
Telephone: 416-231-2383 Voice
416-231-9202 BYTEnet System

Vernon Paige
EPSNLINK
3 McNicoll Avenue
Willowdale, Ontario, Canada M2H 2A6
300 & 1200 Baud
Telephone: 416-494-1380 Voice
416-635-9600 BBS

Terry Smythe
Sysop, Z-Node 40
Muddy Water User Group
55 Rowand Avenue
Winnipeg, Manitoba, Canada R3J 2N6
Telephone: 204-832-3982 Voice
204-945-6713 Voice
204-832-4593 BBS

Denmark:

Beverly Kleiman
International Representative
Personal Computer Society of Denmark
Kronprinsensgade 14,
DK-1114 Copenhagen, Denmark
300 Baud, CCITT Standard
Telephone: 01-122518 BBS

England:

Frank Thornley
67 Woodbridge Road,
Guildford,
Surrey GU21 1JP, United Kingdom
CompuLink
Telephone: 0-483-65895 Voice
0-483-573337 (300/1200 Baud) BBS
0-483-573338 (1200/2400 Baud) BBS

Finland:

Juha Wiio
Databox Oy
Museokatu 11
00100 Helsinki, Finland
DATABOX FIDO
300 & 1200 Baud
Telephone: 358-0-497904

Vivian Ronald Dwight
Suvikuja 3 B 14
02120 Espoo, Finland
Micro Maniacs III Fido Node 17
300 & 1200 & 2400 Baud
Telephone: 358-0-424524 Voice
358-0-4557307 Voice
358-0-467673 BBS

France:

Bill Graham,
President
OUF! (Ordinateurs Utilisateurs France)
ATTN: OUFLOG, B.P. 62
10 rue Saint Nicolas
75012 Paris, France
300 Baud, CCITT Standard
Telephone: 331-43-44-06-48 Voice (Bill)
331-43-44-82-65 Voice
331-43-41-61-47 OUFLOG
for BYTE Listings
331-43-40-33-79 OUFTEL
300 & 1200 Baud
331-43-07-95-39 OUFTEL

Dr. Bernard Pidoux
Groupe Des Utilisateurs Francophones D'Informatique
37, Boulevard Saint-Jacques
75014 Paris, France
300 Baud, CCITT Standard
Telephone: 1-47-63-72-50 Voice
1-45-65-10-09 GUFINET
1-45-65-10-11 GUFITEL

Hong Kong:

W. A. Hanafi
SEAnet
Suite 812, Star House,
Tsim Sha Tsui, Kowloon, Hong Kong
ATTN: Christine Wong
Telephone: 5-455088 Voice
5-8937856 SEAnet 1
5-724495 SEAnet 2

Indonesia:

James D. Filgo
US Embassy Box R
APO SF 96356-5000
Jakarta Computer Society
300 Baud, Bell & CCITT Standard
Telephone: 062-21-799-3286 BBS

Ireland:

Gerry Clarke
30 Auburn Road
Dunlaoire County, Dublin, Ireland
Dublin Bay Bulletin Board
300 & 1200 Baud
Telephone: 353-01-854179

Italy:

Bruno Bonino
MICRO design s.r.l.
Via Rostan, 1
16155 Genova, Italy
C.B.B.S.
CCITT & Bell Standard
Telephone: 10-687098 Voice
10-688783 BBS

Giorgio Leo Rutigliano
Via degli Oleandri, 7
POB 175
85100-Potenza, Italy
FIDO-PZ
300 Baud
Telephone: 0971-34593 Voice (Work)
0971-54431 Voice (Home)
0971-35447 BBS

Claudio Vandelli
Amministratore Unico
SOFT SERVICE s.r.l.
Via G. B. Morgagni 32
20129 Milano, Italy
SOFT SERVICE BBS
300 Baud, CCITT Standard, 8 Bits, No Parity, Full Duplex
Telephone: 02-209231 Voice
02-228467 BBS

Paolo Marraffa
Computronix
Via De Amicis 76
90145 Palermo, Italy
Network Computer Club
300 Baud, CCITT Standard, 8 Bits, 1 Stop Bit, Full Duplex
Telephone: 39-91-266021 BBS
39-91-300229 BBS

Japan:

Peter Perkins
Vice President
Honda Trading Company Ltd.
Mail 101
9-91-Chome, Sota Kanda
Chiyoda-ku, Tokyo, Japan
JANIS
300 & 1200 Baud, CCITT Standard
Telephone: 03-251-0855 BBS

Malaysia:

Ong Boo Huat
3, Jalan Pisang
Jalan Kelang Lama, 58000 Kuala Lumpur
STARLINK
300 Baud
Telephone: 03-7578811 X 116 Voice
03-7576644 BBS

Nigeria:

Chester W. Vlaun
MTCE/31
POB 263
Port Harcourt, Nigeria, West Africa
300 Baud
Telephone: 234-84-301210 to 301229-3022

Norway:

Robert Hertz
Hertz Data Inc.
Huitssfeldts Gate 16
N-0253 Oslo, Norway
Hacker's Unlimited
Telephone: 47-2-431655 Voice
47-2-390521 BBS

Helge Vindenes
5670 FUSA, Norway
Costa del
Telephone: 47-5-151610 Voice
47-5-234129 BBS

Saudi Arabia:

Larry Layland
System Operator DPCS
Aramco
Box 10063
Dhahran, Saudi Arabia 31311
Dhahran Personal Computing Society Bulletin Board
Telephone: 03-873-7851 BBS

Singapore:

Ken Ong
10 Orange Grove Road
#04-01
Singapore 1025, Singapore
K.B.B.S.
300 & 1200 Baud
Telephone: (IDD) 65-734-5825 Voice
(IDD) 65-737-4090 BBS

Sweden:

Jacob Palme
Stockholm University Computer Centre-QZ
Box 27322
102 54 Stockholm, Sweden
BYTECOM
Telephone: 46-8-65-45-00 Voice (Work)
08-23-86-60 (300 Baud)
08-23-89-30 (300 Baud)
08-15-59-20 (300 Baud)
08-14-35-00 (1200 Baud)
08-22-81-30 (1200 Baud)
08-24-61-20 (1200 Baud)
08-14-53-70 (1200 Baud)

Carl Nordin
Nyakersgatan 8B
531 41 Lidkoping, Sweden
A.T.L.
300 & 1200 Baud, CCITT Standard
Telephone: 46-510-25280 Voice
46-510-20409 BBS

Switzerland:

Peter M. C. Werner
9, rue de la Colombiere
1260 Nyon, Switzerland
OCTET
300 & 1200 & 2400 Baud, CCITT Standard
Telephone: 41-22-62-16-54 Voice
41-22-62-18-17 BBS

Albert F. Studer
Technical Director
Kupfer Electronic AG
Soodstrasse 53
Postfach, 8134 Adliswil, Switzerland
TRAX
300 Baud, CCITT Standard
Telephone: 01-710-81-11 Voice
01-710-44-36 BBS

The Netherlands:

Henk Wevers
Cloeckendaal 38
6715 GH Ede, The Netherlands
Henk Wevers' Fido
Telephone: 31-8380-37156 BBS

West Germany:

Rupert Mohr
RMI Nachrichtentechnik GmbH
RosstraBe 7
Postfach 1526
D-5100 Aachen, West Germany
RMI Net
Telephone: 49-241-21145 Voice
45-2410-90528 BYTEnet - DATEX-P
0-26245-2410-90528 User Data - DATEX-P

Rudolf Stricker
Utzoeldstr. 20
D-8000 Munich 22, West Germany
T-BUS FIDO
Telephone: 089-29-38-81 BBS

1987–88 EDITORIAL CALENDAR

AUGUST
Prolog: A look at logic programming with articles on tips and techniques and explorations of the tasks Prolog is best suited for.

SEPTEMBER
Printer Technologies: An examination of the state of the art in printer technologies, including laser, liquid-crystal shutter, and ink-jet technologies.

OCTOBER
Heuristic Algorithms: Artificial intelligence techniques for giving computers the ability to learn from experience.

NOVEMBER
High-Performance Workstations: A tour of the technology underlying the workstations used by scientists and engineers in computer-aided engineering/design.

DECEMBER
Natural Language Processing: The technology of getting computers to understand the natural language of man.

JANUARY
Managing Megabytes: Looking at the ways computers store and retrieve data in situations where disk space is measured in gigabytes and memory is measured in megabytes. Also a look at the new applications that mega-memory and storage will permit.

FEBRUARY
LISP: A BYTE reexamination of the original language of artificial intelligence research.

MARCH
Floating-Point Processors: A look at the processors that speed the computation of mathematical operations in personal computers, including coprocessors and array processors.

APRIL
Memory Management: The hardware and software issues in managing a personal computer's memory space.

MAY
CPU Architectures: An exploration of the latest 32-bit microprocessors, including digital signal processors and programmable graphics processors.

DISKS AND DOWNLOADS

ORDERING DISKS OF BYTE LISTINGS

Listings that accompany BYTE articles are available in a variety of disk formats and on Cauzin Softstrip. Each disk package (which sometimes consists of more than one disk) contains an entire month's listings. If you want to order a disk package from a previous month, please call (603) 924-9281 to find out how many disks it includes. To order listings (for noncommercial use only), fill out this form and send a check or money order in the correct amount to:

BYTE Listings
One Phoenix Mill Lane
Peterborough, NH 03458

All prices include postage. Program listings can also be downloaded via BYTEnet Listings at (617) 861-6764.

BYTE issue:_____

CP/M STANDARD 8-INCH FORMAT
All cost $9.95, $11.95 outside U.S.A.
Annual subscription is $79.95, $99.95 outside U.S.A.

COMMON 5¼-INCH FORMATS
All cost $8.95, $10.95 outside U.S.A.
Annual subscription is $69.95, $89.95 outside U.S.A.

- ☐ Apple II
- ☐ IBM PC
- ☐ Kaypro 2 CP/M
- ☐ MS-DOS 8 Sector
- ☐ Texas Instruments Professional
- ☐ TRS-80 Model 4

COMMON 3½-INCH FORMATS
All cost $9.95, $11.95 outside U.S.A.
Annual subscription is $79.95, $99.95 outside U.S.A.

- ☐ Apple Macintosh
- ☐ Atari 520ST
- ☐ Amiga
- ☐ Hewlett-Packard 150
- ☐ IBM PS/2

SEND TO:

Name _____
Street _____
City_____ State or Province_____
Postal Code _____ Country _____
Check or money order enclosed for $_____

Announcing BYTE's New Subscriber Benefits Program

Your BYTE subscription brings you a complete diet of the latest in microcomputer technology every 30 days. The kind of broad-based objective coverage you read in every issue. *In addition*, your subscription carries a wealth of other benefits. Check the check list:

DISCOUNTS

- ☑ 13 issues instead of 12 if you send payment with subscription order.
- ☑ One-year subscription at $21 (50% off cover price).
- ☑ Two-year subscription at $38.
- ☑ Three-year subscription at $55.
- ☑ One-year GROUP subscription for ten or more at $17.50 each. (Call or write for details.)

SERVICES

- ☑ ***BIX:*** BYTE's Information Exchange puts you on-line 24 hours a day with your peers via computer conferencing and electronic mail. All you need to sign up is a microcomputer, a modem, and telecomm software.
- ☑ ***Reader Service:*** For information on products advertised in BYTE, circle the numbers on the Reader Service card enclosed in each issue that correspond to the numbers for the advertisers you select. Drop it in the mail and we'll get your inquiries to the advertisers.
- ☑ ***TIPS:*** BYTE's Telephone Inquiry System is available to subscribers who need *fast response*. After obtaining your Subscriber I.D. Card, dial TIPS and enter your inquiries. You'll save as much as ten days over the response to Reader Service cards.

- ☑ ***Disks and Downloads:*** Listings of programs that accompany BYTE articles are now available free on the BYTEnet bulletin board, and on disk or in quarterly printed supplements.
- ☑ ***Microforms:*** BYTE is available in microform from University Microfilm International in the U.S. and Europe.
- ☑ ***BYTE's BOMB:*** BYTE's Ongoing Monitor Box is your direct line to the editor's desk. Each month, you can rate the articles via the Reader Service card. Your feedback helps us keep up to date on your information needs.
- ☑ ***Customer Service:*** If you have a problem with, or a question about, your subscription, you may phone us during regular business hours (Eastern time) at our toll-free number: 800-258-5485. You can also use Customer Service to obtain back issues and editorial indexes.

BONUSES

- ☑ ***Annual Separate Issues:*** In addition to BYTE's 12 monthly issues, subscribers also receive our annual IBM PC issue free of charge, as well as any other annual issues BYTE may produce.
- ☑ ***BYTE Deck:*** Subscribers receive five BYTE postcard deck mailings each year—a direct response system for you to obtain information on advertised products through return mail.

To be on the leading edge of microcomputer technology *and* receive all the aforementioned benefits, make a career decision today. Call toll-free weekdays, 8:30am to 4:30pm Eastern time: 800-258-5485.

And . . . welcome to BYTE country!

BYTE
THE SMALL SYSTEMS JOURNAL

McGraw Hill

Borland's new Turbo C: The most powerful optimizing compiler ever

Our new Turbo C® generates fast, tight, production-quality code at compilation speeds of more than 13,000* lines a minute!

It's the full-featured optimizing compiler everyone has been waiting for.

Switching to Turbo C, or starting with Turbo C, you win both ways

If you're already programming in C, switching to Turbo C will make you feel like you're riding a rocket instead of pedaling a bike.

If you've never programmed in C, starting with Turbo C gives you an instant edge. It's easy to learn, easy to use, and the most efficient C compiler at any price.

"Turbo C does look like What We've All Been Waiting For: a full-featured compiler that produces excellent code in an unbelievable hurry . . . moves into a class all its own among full-featured C compilers . . . Turbo C is indeed for the serious developer . . . One heck of a buy—at any price."

Michael Abrash, Programmer's Journal

Join more than 100,000 Turbo C enthusiasts. Get your copy of Turbo C today!

Technical Specifications

- Compiler. One-pass optimizing compiler generating linkable object modules. Included is Borland's high-performance Turbo Linker.™ The object module is compatible with the PC-DOS linker. Supports tiny, small, compact, medium, large, and huge memory model libraries. Can mix models with near and far pointers. Includes floating point emulator (utilizes 8087/80287 if installed).
- Interactive Editor. The system includes a powerful, interactive full-screen text editor. If the compiler detects an error, the editor automatically positions the cursor appropriately in the source code.
- Development Environment. A powerful "Make" is included so that managing Turbo C program development is highly efficient. Also includes pull-down menus and windows.
- Links with relocatable object modules created using Borland's Turbo Prolog® into a single program.
- Inline assembly code.
- Loop optimizations.
- Register variables.
- ANSI C compatible.
- Start-up routine source code included.
- Both command line and integrated environment versions included.
- License to the source code for Run-time Library available.

Sieve benchmark

	Turbo C	Microsoft® C
Compile time	2.4	13.51
Compile and link time	4.1	18.13
Execution time	3.95	5.93
Object code size	239	249
Execution size	5748	7136
Price	$99.95	$450.00

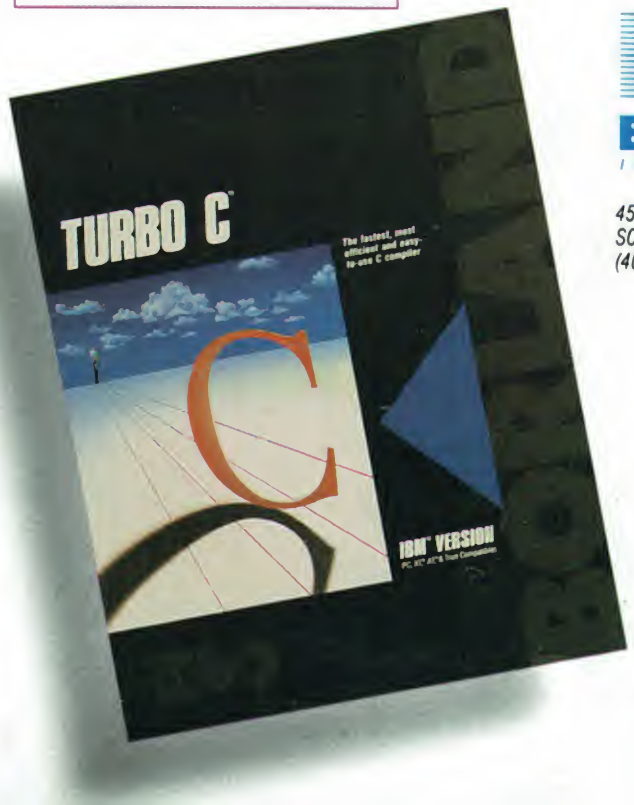
*Benchmark run on an IBM PS/2 Model 60 using Turbo C version 1.0 and the Turbo Linker version 1.0; Microsoft C version 4.0 and the MS overlay linker version 3.51.

Minimum system requirements: IBM PC, XT, AT, PS/2 and true compatibles PC-DOS (MS-DOS) 2.0 or later. 384K.

For the dealer nearest you or to order by phone call

(800) 255-8008

in CA (800) 742-1133 in Canada (800) 237-1136



BORLAND INTERNATIONAL

4585 SCOTTS VALLEY DRIVE
SCOTTS VALLEY, CA 95066
(408) 438-8400 TELEX: 172373

Only $99.95!